НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТЕТУТ»

Кафедра обчислювальної технік

**Курсовий проект**

з дисципліни «Бази даних»

на тему : Адміністратор електронної системи «Амбулаторія»

Студента III курсу групи ІП-22

Напрямку підготовки 6.050103

Спеціальності «Програмна інженерія»

Борисов О.В.

Керівник к.т.н, доцент Болдак А.О.

Національна оцінка

Кількість балів : Оцінка : ECTS

(Підпис) (Вчение звання, науковий ступінь, прізвище та ініціали)

(Підпис) (Вчение звання, науковий ступінь, прізвище та ініціали)

Київ - 2014

# ЗМІСТ

[РОЗДІЛ 1 1](#_Toc404636262)

[Запити зацікавлених осіб 1](#_Toc404636263)

[2.1. Введення 1](#_Toc404636264)

[2.2. Короткий огляд продукту 1](#_Toc404636265)

[2.3. Ділові правила та приписи 2](#_Toc404636266)

[2.4. Функціональність 5](#_Toc404636267)

[2.5. Можливості адміністраторів 5](#_Toc404636268)

[2.6. Практичність 5](#_Toc404636269)

[2.7. Надійність 5](#_Toc404636270)

[РОЗДІЛ 2 6](#_Toc404636271)

[ПРЕЦЕДЕНТИ 6](#_Toc404636272)

[2.1. Загальна схема прецедентів 6](#_Toc404636273)

[2.2. Прецеденти для ролі адміністратора 6](#_Toc404636274)

[2.3. Діаграма бізнес-сутностей 15](#_Toc404636275)

[2.4. Реляційна модель бази даних 16](#_Toc404636276)

[2.5. Специфікація таблиць бази даних 16](#_Toc404636277)

[РОЗДІЛ 3 20](#_Toc404636278)

[РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ 20](#_Toc404636279)

[3.1. Реляційно-об’єктне відображення 20](#_Toc404636281)

[3.2. Діаграма DAO класів 24](#_Toc404636282)

[3.3. Класи сервелетів та їх специфікація 48](#_Toc404636283)

[3.3.1 Класс AuthorizeServlet 48](#_Toc404636284)

[РОЗДІЛ 4 53](#_Toc404636285)

[ІЛЮСТРАЦІЯ РОБОТИ ПРОГРАМИ 53](#_Toc404636286)

[4.1. Ілюстрація сценарію «Входу в систему» 53](#_Toc404636288)

[4.2. Ілюстрація сценарію «Перегляд користувачів» 55](#_Toc404636289)

[4.3. Ілюстрація сценарію «Перегляд карток» 56](#_Toc404636290)

[4.4. Ілюстрація сценарію «Створення нового користувача» 57](#_Toc404636291)

[4.5. Ілюстрація сценарію «Додати хворобу» 59](#_Toc404636292)

[4.6. Ілюстрація сценарію «Видалення хвороби» 61](#_Toc404636293)

[4.7. Ілюстрація сценарію «Редагування власних даних» 62](#_Toc404636294)

[4.8. Ілюстрація сценарію «Редагування користувача» 63](#_Toc404636295)

[4.9. Ілюстрація сценарію «Редагування картки хворого» 64](#_Toc404636296)

[СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ 65](#_Toc404636297)

[ДОДАТКИ 66](#_Toc404636298)

[ДОДАТОК А 66](#_Toc404636299)

[ДОДАТОК Б 67](#_Toc404636300)

[ДОДАТОК В 70](#_Toc404636301)

# РОЗДІЛ 1

# Запити зацікавлених осіб

## Введення

У цьому документі описуються запити зацікавлених осіб, в якості яких виступає замовник – Болдак А.А. , по відношенню до системи обліку пацієнтів для медичного закладу(далі система «Амбулаторія»).

### Ціль

Ціллю документу є визначення основних вимог до функціональності, продуктивності та експлуатаційної придатності, а також визначення бізнес-правил і технічних обмежень до предмету розробки.

### Контекст

Перелік вимог, що перераховуються в даному документі, є основною технічного завдання на розробку системи «Амбулаторія».

## Короткий огляд продукту

Система «Амбулаторія» дозволяє будь-якому медичному закладу (за умови наявності апаратного забезпечення) вести повний облік пацієнтів даного медичного закладу, відслідковувати процес лікування хворого на основі історії хвороби, аналізів та інших даних, які працівники медичного закладу можуть додавати до карти пацієнта, забезпечує взаємозв’язок між пацієнтами та лікарями.

## Ділові правила та приписи

### Призначення системи «Амбулаторія»

Функціональність системи дозволяє полегшити лікарям слідкувати за процесом лікування своїх хворих, вести історію хвороби, формувати виписки, рецепти, лікарські форми. Забезпечує «прозорість» та зручність лікувального процесу за допомогою зворотнього зв’язку пацієнтів та лікарів та можливості перегляду карти хворого самим пацієнтом на своїй персональній сторінці. Зв’язок лікарів зі своїми колегами та обмін даними між різними медичними закладами забезпечить кращу якість лікування.

### Політика взаємовідносин з клієнтом

Клієнтом системи «Амбулаторія» можуть бути як і фізичні особи — працівники медичного закладу та будь-які інші особи, які перебувають чи перебували на лікуванні, так і юридичні — виробники ліків чи інших лікарських засобів, інші медичні заклади, тощо.

### Характеристика ділового процесу

Управління системою «Амбулаторія» виконує адміністратор, який відповідає за дієздатність усієї системи та проведення операцій з аккаунтами мед працівників та пацієнтами.

Мед працівники мають доступ до історії хвороб пацієнтів, які дозволені їм для перегляду. Також вони можуть створювати, редагувати, закривати історії хвороби пацієнтів, надсилати їм повідомлення(листування, направлення, рецепти та інш.).

### Сценарій реєстрації нового мед працівника

Передумова – адміністратору приходять дані для реєстрації нового працівника.

1. Адміністратор натискає кнопку «Зареєструвати нового працівника».
2. Система відкриває форму «Реєстрації нового працівника».
3. Адміністратор заповнює поля: П.І.Б., пароль, перевірка паролю, документ затвердження працівника, галузь діяльності, номер телефону, адресу, e-mail.
4. Система перевіряє дані, порівнює з поточними аккаунтами для виявлення однакових. І відповідно до результату реагує.
5. Якщо проблем не виникло адміністратор може підтвердити дані.
6. Система сповіщає про внесення працівника і відправляє повідомлення йому про це на e-mail.

### Сценарій редагування аккаунта працівника

1. Вповноважена особа(адміністратор, володар аккаунта) натискає «Редагувати особисті дані».

2. Система відкриває форму «Редагування особистих даних працівника».

3. Вповноважена особа заповнює поля: пароль, перевірка паролю, номер телефону, адресу, e-mail.

4. Вповноважена особа може підтвердити дані.

5. Система сповіщає про внесення змін і відправляє повідомлення на e-mail.

### Сценарій створення нової історії хвороби

1. Мед працівник лікар водить в поле «Пацієнти» П.І.Б. потрібного пацієнта.

2. Система відкриває картку пацієнта.

3. Працівник натискає кнопку «Створити нову історію хвороби».

4. Система відкриває форму «Створення нової історії хвороби».

5. Працівник заповнює відповідні поля: (необхідне уточнення). І нитискає кнопку «Зберегти».

6. Працівник підтверджує внесення даних.

### Сценарій створення нової історії хвороби

1. Мед працівник лікар водить в поле «Пацієнти» П.І.Б. потрібного пацієнта.

2. Система відкриває картку пацієнта.

3. Працівник натискає кнопку «Закрити історію хвороби».

4. Система відкриває форму «Підтвердження».

5. Працівник підтверджує зміни натиснувши кнопку «Так».

6. Система вносить зміни.

### Сценарій внесення даних в історію хвороби

1. Мед працівник лікар водить в поле «Пацієнти» П.І.Б. потрібного пацієнта.

2. Система відкриває картку пацієнта.

3. Працівник обирає потрібну історію.

4. Система відкриває історію пацієнта.

5. Працівник натискає «Ввнести дані».

6. Працівник водить дані та нажимає «Зберегти».

7. Система заносить дані та сповіщає про це.

## Функціональність

Основні вимоги до функціоналу, пред’явлені зацікавленою особою до предмету розробки, відносяться:

* Адміністратори.

## Можливості адміністраторів

- Створення акаунта мед працівника;

- Видалення акаунта мед працівника;

- Видалення картки клієнта;

- Редагування акаунта мед працівника;

- Редагування картки клієнта;

- Видалення акаунта адміністратора;

- Внесення нової хвороби.

## Практичність

Веб-сервіс повинен підтримувати українську мову.

## Надійність

Повинно здійснюється резервне копіювання баз даних.

# РОЗДІЛ 2

# ПРЕЦЕДЕНТИ

## **Загальна схема прецедентів**

Загальна схема прецедентів для ролі адміністратора показує можливі послідовності дій актора. У даному випадку Адміністратор має 3 основні види діяльності: редагування аккаунтів користувачів, редагування карток хворих та редагування списку хвороб. Більш детально розкриває діаграма (рис. 2.1).

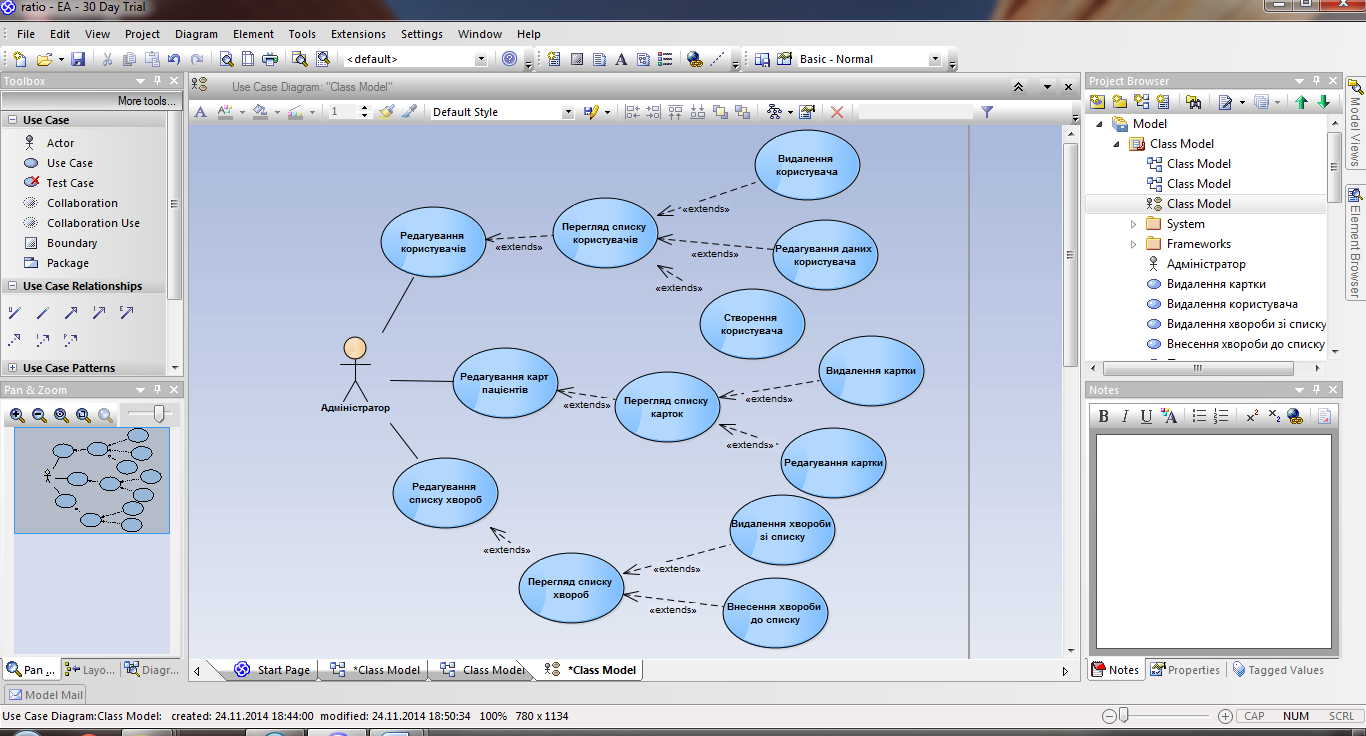


Рис. 2.1 Загальна схема прецедентів для ролі адміністратора

## Прецеденти для ролі адміністратора

Нижче описані прецеденти для ролі адміністратора зі вказаними передумовами, результатом, виключними ситуаціями та детально розписаним основним сценарієм.

UseCase01. Створення нового профілю працівника

Таблиця 2.1

|  |  |  |  |
| --- | --- | --- | --- |
| ID | А01 | | |
| Назва | Створення нового профілю користувача | | |
| Учасники | Адміністратор, Система | | |
| Передумова | - | | |
| Результат | Новий профіль користувача в системі | | |
| Основний сценарій |  | Адміністратор | Система |
| 1 | Натискає кнопку «Створення нового користувача» |  |
| 2 |  | Показує форму створення нового користувача |
| 3 | Заповнює форму |  |
| 4 | Натискає кнопку «зареєструвати» |  |
| 5 |  | Перевіряє введені адміністратором та  додає новий профіль користувача |
| Виключні ситуації | Неправильно введені дані | | |
| Сутність | Адміністратор, Система, форма створення профілю нового користувача | | |

UseCase02. Створення нової карточки

Таблиця 2.2

|  |  |  |  |
| --- | --- | --- | --- |
| ID | А02 | | |
| Назва | Створення нової медкартки | | |
| Учасники | Модератор, Система | | |
| Передумова | Модератор авторизувався | | |
| Результат | Новий мед картка в системі | | |
| Основний сценарій |  | Модератор | Система |
| 1 | Натискає кнопку «Створення нової медкартки» |  |
| 2 |  | Показує форму створення нової медкартки |
| 3 | Заповнює форму |  |
| 4 | Натискає кнопку «зберегти» |  |
| 5 |  | Перевіряє введені модератором дані та  додає нову медкартку в базу даних |
| Виключні ситуації | Неправильно введені дані | | |
| Сутність | Модератор, Система, форма створення нової медкартки | | |

UseCase03. Внесення змін до карточки хворого

Таблиця 2.3

|  |  |  |  |
| --- | --- | --- | --- |
| ID | А03 | | |
| Назва | Внесення змін до медкартки хворого | | |
| Учасники | Модератор, Система | | |
| Передумова | Медкартка існує | | |
| Результат | Зміна даних в медкартці | | |
| Основний сценарій |  | Модератор | Система |
| 1 | Відкриває медкартку хворого |  |
| 2 |  | Виводить медкартку для перегляду та редагування( тільки доступні поля) |
| 3 | Вносить зміни в доступні йому поля |  |
| 4 | Натискає кнопку «зберегти» |  |
| 5 |  | Перевіряє введені модератором дані та вносить зміни в базу даних |
| Виключні ситуації | Неправильно введені дані | | |
| Сутність | Модератор, Система, медкартка | | |

UseCase04. Перегляд карточки хворого

Таблиця 2.4

|  |  |  |  |
| --- | --- | --- | --- |
| ID | А07 | | |
| Назва | Перегляд карточки хворого | | |
| Учасники | Користувач, Система | | |
| Передумова | Карточка хворого існує | | |
| Результат | Користувач переглядає визначені дані в карточці | | |
| Основний сценарій |  | Дії Користувача | Дії Системи |
| 1 | Обирає потрібну медичну карточку |  |
| 2 | Натискає кнопку «Переглянути» |  |
| 3 |  | Відкриває карточку для перегляду із урахуванням доступу користувача |
| Виключні ситуації |  | | |
| Сутності | Користувач, Система, Медична Картка | | |

UseCase05. Зміна даних профілю працівника

Таблиця 2.5

|  |  |  |  |
| --- | --- | --- | --- |
| ID | А05 | | |
| Назва | Зміна даних профілю працівника | | |
| Учасники | Адміністратор, система | | |
| Передумова | Потрібно змінити дані певного користувача | | |
| Результат | Зміна даних профілю відповідного користувача | | |
| Основний сценарій |  | Дії Адміністратора | Дії Системи |
| 1 | Обираю профіль потрібного користувача |  |
| 2 | Натискає кнопку «Редагування» |  |
| 3 |  | Надає відповідну форму |
| 4 | Редагує потрібні поля в профілі |  |
| 5 | Натискає кнопку «Зберегти» |  |
| 6 |  | Перевірка правильності введених даних |
| 7 |  | Виводить повідомлення про успішне збереження змін |
| Виключні ситуації | Неправильно введені дані | | |
| Сутності | Адміністратор, Система, Форма редагування | | |

UseCase06. Внесення даних про оплату курсу лікування

Таблиця 2.6

|  |  |  |  |
| --- | --- | --- | --- |
| ID | A06 | | |
| Назва | Внесення даних про оплату курсу лікування | | |
| Учасники | Бухгалтер, Система | | |
| Передумова | Пацієнт отримав рахунок | | |
| Результат | Внесення даних про оплату послух амбулаторії | | |
| Основний сценарій |  | Дії Бухгалтера | Дії Системи |
| 1 | Обирає відповідного медичну карточку |  |
| 2 | Обирає послугу за яку виконується оплата |  |
| 3 | Встановлює checkBox «Оплачено» активним |  |
| 4 |  | Запам’ятовую активацію checkBox |
| Виключні ситуації | Бухгалтер натиснув не на той checkBox | | |
| Сутності | Бухгалтер, Система | | |

UseCase07. Вхід до системи

Таблиця 2.7

|  |  |  |  |
| --- | --- | --- | --- |
| ID | А08 | | |
| Назва | Вхід до системи | | |
| Учасники | Система, Неавторизований користувач | | |
| Передумова |  | | |
| Результат | Неавторизований користувач стає авторизованим | | |
| Основний сценарій |  | Дії Неавторизованого користувача | Дії Системи |
| 1 | Натискає кнопку «Вхід» |  |
| 2 |  | Надає форму для входу в систему |
| 3 | Заповнює поля наданої форми |  |
| 4 |  | Перевіряє дані |
| 5 |  | Переводить неавторизованого користувача у статус авторизованого |
| Виключні ситуації | Дані введені не правильно | | |
| Сутності | Система, Неавторизований користувач | | |

UseCase08. Пошук карточки хворого

Таблиця 2.8

|  |  |  |  |
| --- | --- | --- | --- |
| ID | А08 | | |
| Назва | Пошук медкартки | | |
| Учасники | Користувач, Система | | |
| Передумова | Користувач авторизувався | | |
| Результат | Вивід медкарток що відповідають запиту користувача | | |
| Основний сценарій |  | Користувач | Система |
| 1 | Натискає «Пошук» |  |
| 2 |  | Система відкриває форму параметрів пошуку |
| 3 | Виставляє параметри пошуку |  |
| 4 |  | Виконує перевірку всіх медкарток на відповідність пошукового запиту користувача і в разі позитивного результату виводить користувачу |

## Діаграма бізнес-сутностей

Дана діаграма використовується на етапі Бізнес Моделювання. Вона відібражає основні бізнес-сутності та взаємозв’язки між ними. В даному випадку основими сутностями є аккаунт, ставка, подія, команда та ліга, які взаємодіють між собою та включають у себе інші бізнес-сутності. Діаграма бізнес сутностей проекту зображена на рис.2.2.

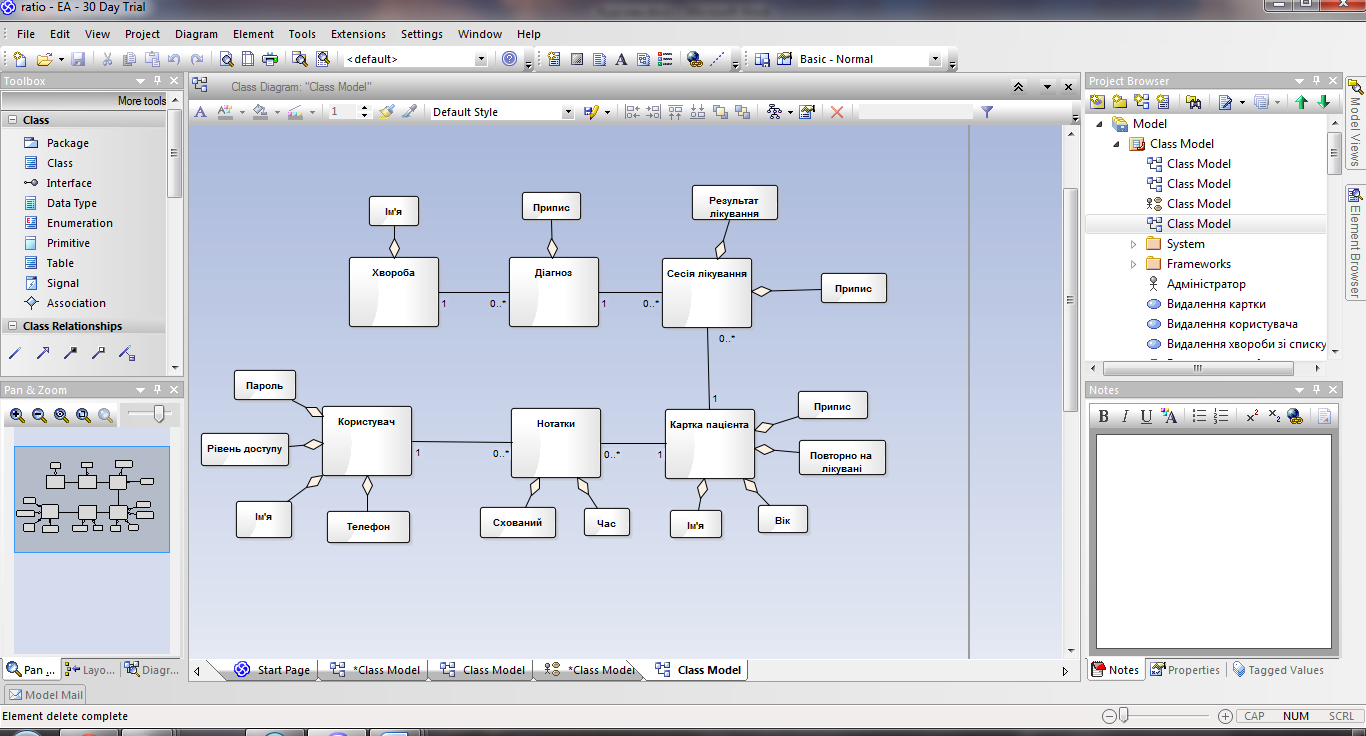


Рис. 2.2 Діаграма бізнес-сутностей

## Реляційна модель бази даних

Реляційна модель бази даних (рис. 2.3) зображає структуру таблиць бази даних, взаємозв’язки між ними та поля кожної з таблиць. Наведена діаграма має багато схожого з діаграмою бізнес-сутностей. Кожній основній бізнес-сутності відповідає таблиця бази даних.

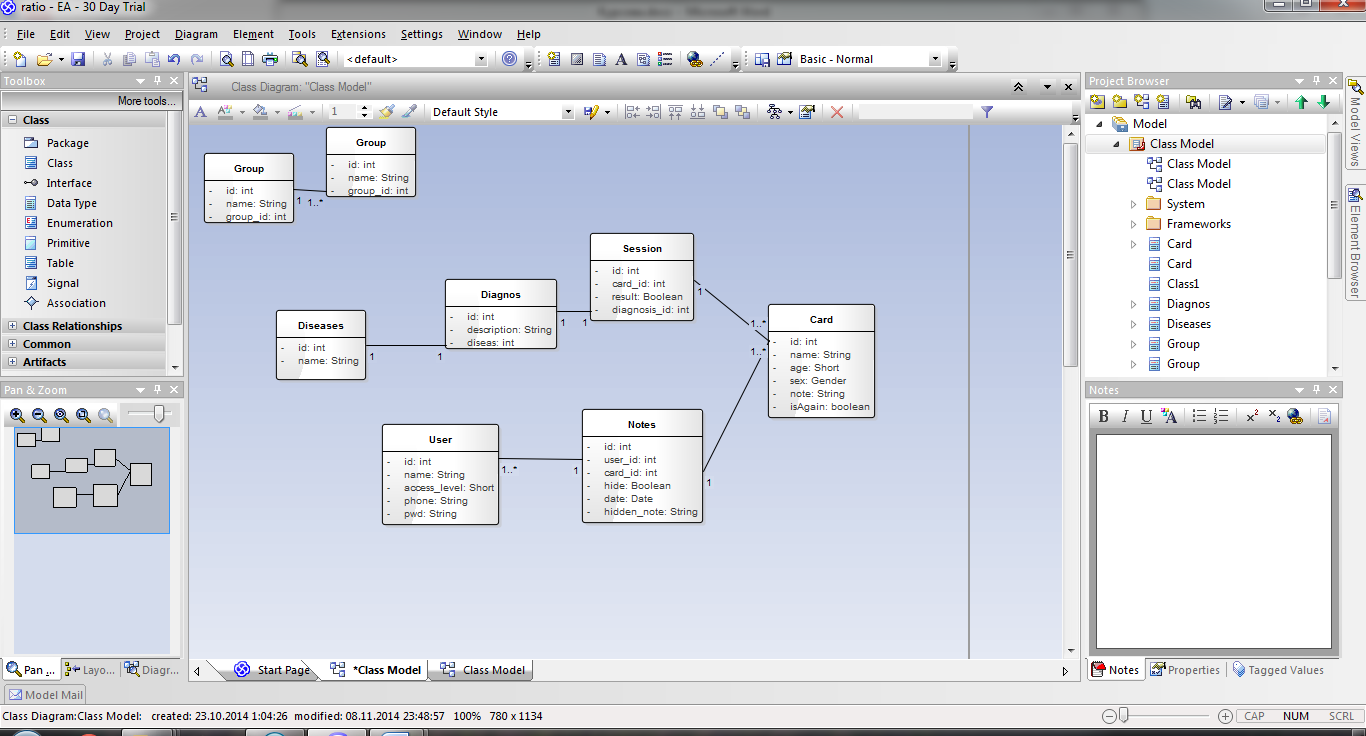


Рис. 2.3 Реляційна модель бази даних

## Специфікація таблиць бази даних

Специфікація таблиць бази даних включає в себе інформацію про назви колонок таблиці, їхній тип, чи є ця колонка первинним ключем, чи поле може бути пустим, чи значення поля автоматично збільшується та коментар щодо призначення цієї колонки. Таблиці зі специфікаціями наведені нижче.

Таблиця «Cards»

Таблиця 2.9

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ім’я | Тип даних | Не пусте | Автоматично  збільшувати | Ключ | Коментарі |
| Id | INTEGER | Так | Так | Так | Індекс в таблиці |
| Name | VARCHAR(45) | Так | Ні | Ні | Ім’я хворого |
| Age | SMALLINT(5) | Так | Ні | Ні | Вік хворого |
| Sex | ENUM(‘male’, female’) | Так | Ні | Ні | Стать хворого |
| Note | TEXT | Ні | Ні | Ні | Припис до картки хворого |
| isAgain | BIT(1) | Так | Ні | Ні | Для визначення чи вже хворий проходив лікування в закладі |

Таблиця «Diagnosis»

Таблиця 2.10

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ім’я | Тип даних | Не пусте | Автоматично  збільшувати | Ключ | Коментарі |
| Id | INTEGER | Так | Так | Так | Індекс в таблиці |
| Description | VARCHAR(100) | Так | Ні | Ні | Припис до діагнозу |
| Disease | INTEGER | Так | Ні | Так | Ключ для витягування хвороби |

Таблиця «Diseases»

Таблиця 2.11

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ім’я | Тип даних | Не пусте | Автоматично  збільшувати | Ключ | Коментарі |
| Id | INTEGER | Так | Так | Так | Індекс в таблиці |
| Name | VARCHAR(45) | Так | Ні | Ні | Ім’я хвороби |

Таблиця «Notes»

Таблиця 2.12

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ім’я | Тип даних | Не пусте | Автоматично  збільшувати | Ключ | Коментарі |
| Id | INT(10) | Так | Так | Так | Індекс в таблиці |
| User\_id | INT(10) | Так | Ні | Так | Ключ для витягування користувача |
| Card\_id | INT(10) | Так | Ні | Так | Ключ для витягування картки |
| Hide | BIT(1) | Так | Ні | Ні | Ховає нотаток від інших |
| Date | DATE | Так | Ні | Ні | Час створення нотатку |
| Hidden\_note | TEXT | Ні | Ні | Ні | Текст нотатку |

Таблиця «Sessions»

Таблиця 2.13

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ім’я | Тип даних | Не пусте | Автоматично  збільшувати | Ключ | Коментарі |
| Id | INTEGER | Так | Так | Так | Індекс в таблиці |
| Card\_id | INTEGER | Так | Ні | Так | Ключ для витягування картки |
| Result | BIT(1) | Так | Ні | Ні | Результат лікування |
| Diagnosis\_id | INTEGER | Так | Ні | Так | Ключ для витягування діагнозу |

Таблиця «Users»

Таблиця 2.14

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ім’я | Тип даних | Не пусте | Автоматично  збільшувати | Ключ | Коментарі |
| Id | INTEGER | Так | Так | Так | Індекс в таблиці |
| Name | VARCHAR(45) | Так | Ні | Ні | Ім’я користувача |
| Access\_level | SMALLINT(6) | Так | Ні | Ні | Рівень доступу користувача |
| Phone | VARCHAR(45) | Ні | Ні | Ні | Телефон користувача |
| pwd | VARCHAR(45) | Так | Ні | Ні | Пароль користувача |

# РОЗДІЛ 3

# РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ



## Реляційно-об’єктне відображення

Для реляційно-об’єктного відображення було використано бібліотеку Hibernate [1]. Hibernate надає можливість легко встановити зв’язок з будь-якою базою даних та створити відображення між об'єктно-орієнтованою моделлю даних і традиційною реляційною базою даних.

На рис. 3.1 зображено діаграму Entity та DAO класів. Детальна специфікація (JavaDoc) наведена нижче.

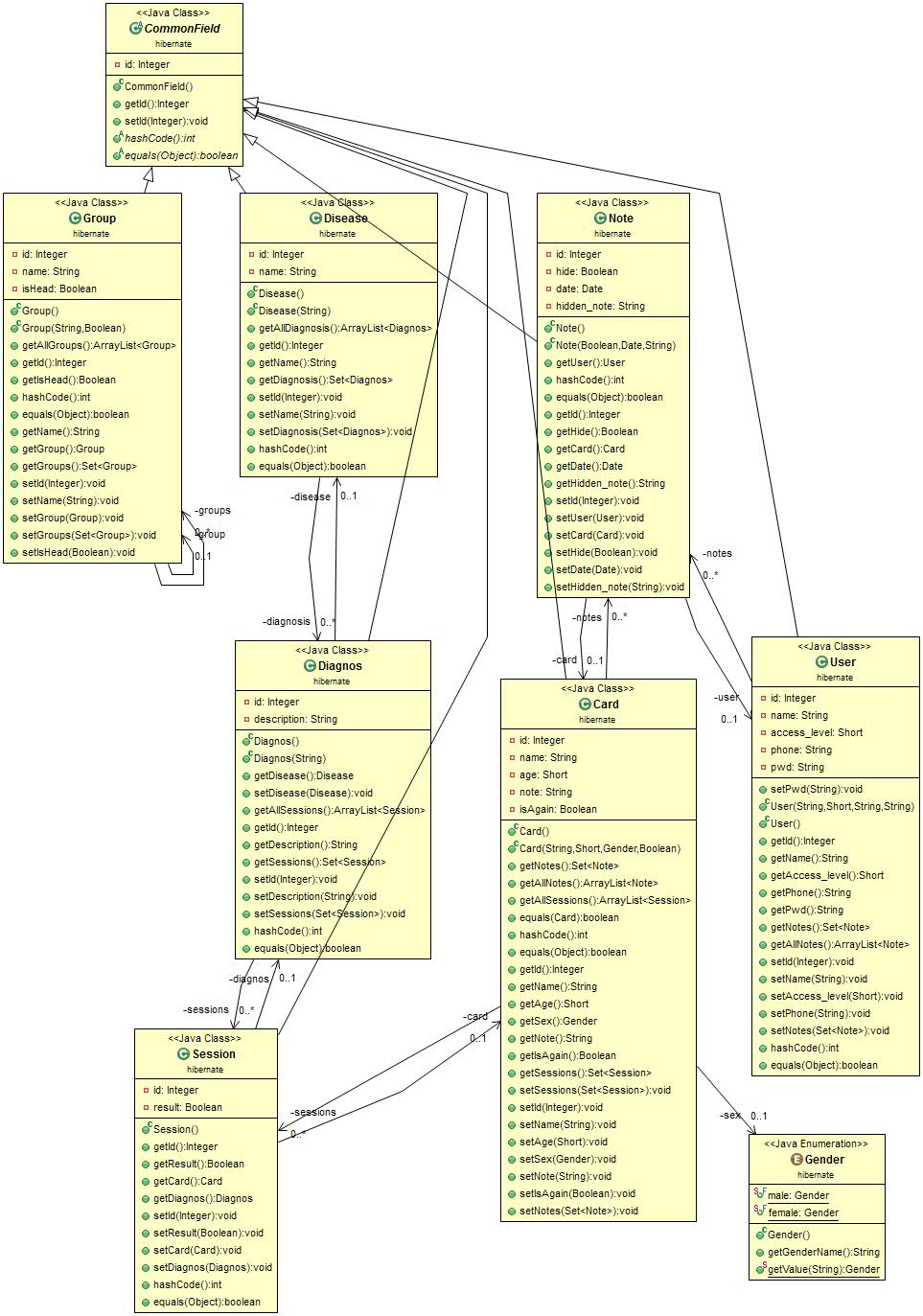


Рис. 3.1. Реляційно-об’єктне відображення

### Клас Card

|  |  |
| --- | --- |
| **Fields** | |
| **Modifier and Type** | **Field and Description** |
| private java.lang.Short | [**age**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html#age)  Опис поля "age" таблиці "cards" |
| private java.lang.Integer | [**id**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html#id)  Опис поля "id" таблиці "cards" |
| private java.lang.Boolean | [**isAgain**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html#isAgain)  Опис поля "isAgain" таблиці "cards" |
| private java.lang.String | [**name**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html#name)  Опис поля "name" таблиці "cards" |
| private java.lang.String | [**note**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html#note)  Опис поля "note" таблиці "cards" |
| private java.util.Set<[**Note**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Note.html)> | [**notes**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html#notes)  Опис зміної масиву даних для витягування всіх нотатків, які стосуються даної картки, із зв'язаної таблиці "sessions" по індефікатору "card\_id", який буде співпадати з індефікатором "id" поля з таблиці "cards". |
| private java.util.Set<[**Session**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Session.html)> | [**sessions**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html#sessions)  Опис зміної масиву даних для витягування всіх сесій, які стосуються даної картки, із зв'язаної таблиці "sessions" по індефікатору "card\_id", який буде співпадати з індефікатором "id" поля з таблиці "cards". |
| private [**Gender**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Gender.html) | [**sex**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html#sex)  Опис поля "sex" таблиці "cards" |

### Клас CommonField

|  |  |
| --- | --- |
| **Fields** | |
| **Modifier and Type** | **Field and Description** |
| private java.lang.Integer | [**id**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\CommonField.html#id)  Індефікатор в таблиці |

### Клас Diagnos

|  |  |
| --- | --- |
| **Fields** | |
| **Modifier and Type** | **Field and Description** |
| private java.lang.String | [**description**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Diagnos.html#description)  Приспис до діагнозу в таблиці "diagnosis" |
| private [**Disease**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Disease.html) | [**disease**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Diagnos.html#disease)  Зв'язний параметр з таблицею "diseases". |
| private java.lang.Integer | [**id**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Diagnos.html#id)  Індефікатор даних в таблиці "diagnosis" |
| private java.util.Set<[**Session**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Session.html)> | [**sessions**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Diagnos.html#sessions)  Опис зміної масиву даних для витягування всіх сесій, які стосуються даного діагнозу, із зв'язаної таблиці "sessions" по індефікатору "diagnosis\_id", який буде співпадати з індефікатором "id" поля з таблиці "diagnos". |

### Клас Disease

|  |  |
| --- | --- |
| **Fields** | |
| **Modifier and Type** | **Field and Description** |
| private java.util.Set<[**Diagnos**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Diagnos.html)> | [**diagnosis**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Disease.html#diagnosis)  Опис зміної масиву даних для витягування всіх діагнозів, які стосуються даної хвороби, із зв'язаної таблиці "diagnosis" по індефікатору "disease", який буде співпадати з індефікатором "id" поля з таблиці "disease". |
| private java.lang.Integer | [**id**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Disease.html#id)  Індефікатор даних в таблиці "diseases" |
| private java.lang.String | [**name**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Disease.html#name)  Назва звороби в таблиці "diseases" |

### Клас Note

|  |  |
| --- | --- |
| **Fields** | |
| **Modifier and Type** | **Field and Description** |
| private [**Card**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html) | [**card**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Note.html#card)  Зв'язний параметр з таблицею "cards". |
| private java.sql.Date | [**date**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Note.html#date)  Поле часу створення нотатку |
| private java.lang.String | [**hidden\_note**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Note.html#hidden_note)  Поле текстового повідомлення нотатку |
| private java.lang.Boolean | [**hide**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Note.html#hide)  Поле для встановлення режиму видемості нотатку |
| private java.lang.Integer | [**id**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Note.html#id)  Індефікатор даних в таблиці "notes" |
| private [**User**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\User.html) | [**user**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Note.html#user)  Зв'язний параметр з таблицею "user". |

### Клас Session

|  |  |
| --- | --- |
| private hibernate.Card | [**card**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Session.html#card)  Зв'язний параметр з таблицею "cards". |
| private hibernate.Diagnos | [**diagnos**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Session.html#diagnos)  Зв'язний параметр з таблицею "diagnosis". |
| private java.lang.Integer | [**id**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Session.html#id)  Індефікатор даних в таблиці "sessions" |
| private java.lang.Boolean | [**result**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Session.html#result)  Результат сесії лікування |

### Клас User

|  |  |
| --- | --- |
| **Fields** | |
| **Modifier and Type** | **Field and Description** |
| private java.lang.Short | [**access\_level**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\User.html#access_level)  Поле містить рівень доступу користувача |
| private java.lang.Integer | [**id**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\User.html#id)  Індефікатор даних в таблиці "users" |
| private java.lang.String | [**name**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\User.html#name)  Поле містить ім'я користувача |
| private java.util.Set<[**Note**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Note.html)> | [**notes**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\User.html#notes)  Опис зміної масиву даних для витягування всіх нотатків, які стосуються даної групи, із зв'язаної таблиці "notes" по індефікатору "user\_id", який буде співпадати з індефікатором "id" поля з таблиці "user". |
| private java.lang.String | [**phone**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\User.html#phone)  Поле містить телефон користувача |
| private java.lang.String | [**pwd**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\User.html#pwd)  Поле містить пароль користувача |

## Діаграма DAO класів

Класи, які тут будуть представлені, необхідні для роботи через Hibernate з базами даних. У методах цих класів формуються запити до бази даних при використані важелів, що надає Hibernate. Діаграму DAO класів можна побачити на рис 3.2.

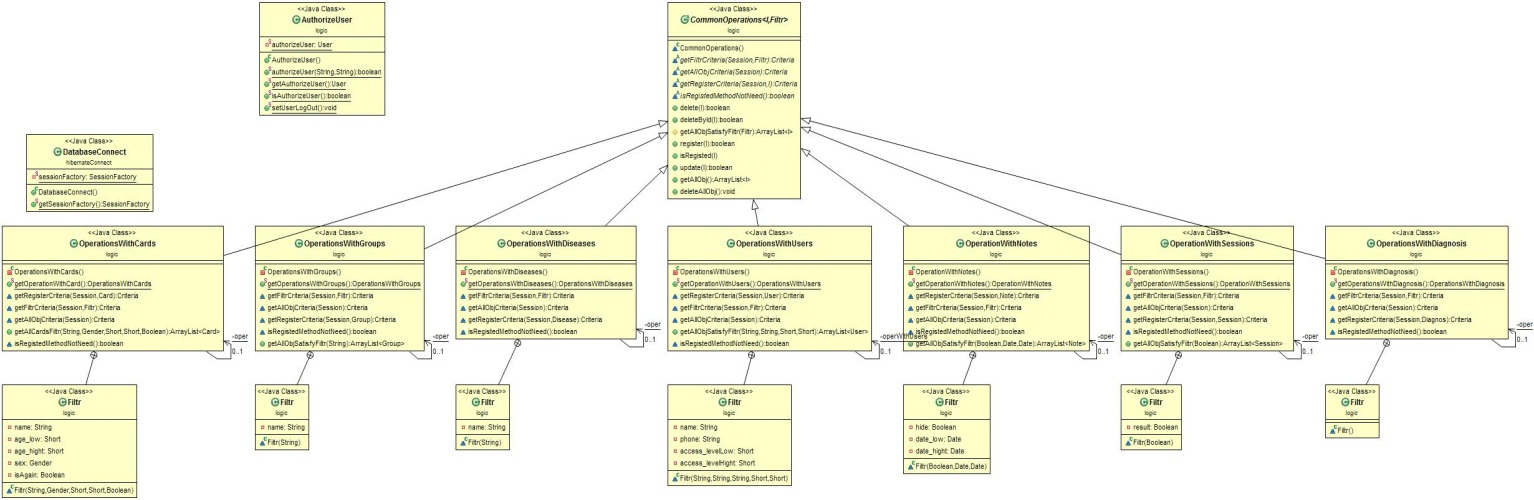


Рис. 3.2 Діаграма DAO класів

### Клас DatabaseConnect

|  |  |
| --- | --- |
| **Fields** | |
| **Modifier and Type** | **Field and Description** |
| private static org.hibernate.SessionFactory | [**sessionFactory**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernateConnect\DatabaseConnect.html#sessionFactory)  Підключення до бази даних |

### *Method Detail*

#### getSessionFactory

public static org.hibernate.SessionFactory getSessionFactory()

**Returns:**

повертає підключення до бази даних

### Клас AuthorizeUser

|  |  |
| --- | --- |
| **Fields** | |
| **Modifier and Type** | **Field and Description** |
| private static [**User**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\User.html) | [**authorizeUser**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\AuthorizeUser.html#authorizeUser)  Авторизовани користувач |

### *Method Detail*

* public static boolean authorizeUser(java.lang.String name,

java.lang.String pwd)

Метод виконує процедуру авторизації

**Parameters:**

name - ім'я користувача

pwd - пароль користувача

**Returns:**

повертає true якщо користувач пройшов авторизацію, повертає false якщо ні

### Клас CommonOperations

### *Method Detail*

#### getFiltrCriteria

* + abstract org.hibernate.Criteria getFiltrCriteria(org.hibernate.Session session,

[Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html) filtr)

Абстрактний метод, який має реалізувати фільтр

**Parameters:**

session - сесія з базою даних

filtr - параметри для фільтрації даних з бази даних

**Returns:**

запит до бази даних для витягування всіх об'єктів, які задовільняють фільтр

#### getAllObjCriteria

abstract org.hibernate.Criteria getAllObjCriteria(org.hibernate.Session session)

Абстрактний метод, який має реалізувати запит до бази даних

**Parameters:**

session - сесія з базою даних

**Returns:**

запит до бази даних для витягування всіх об'єктів класу "I"

#### getRegisterCriteria

* + abstract org.hibernate.Criteria getRegisterCriteria(org.hibernate.Session session,

[I](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html) object)

Абстрактний метод, який реалізує критерії, за якими шукається дані в таблиці

**Parameters:**

session - сесія з базою даних

object - об'єкт, який шукається в базі даних

**Returns:**

запит до бази даних для витягування всіх об'єктів, які задовільняють критерії

#### isRegistedMethodNotNeed

abstract boolean isRegistedMethodNotNeed()

Абстрактний метод, який реалізує використання методу getRegisterCriteria

**Returns:**

повертати true потрібно, коли використання метода getRegisterCriteria не потрібно (збільшує продуктивність), при повернені false метод getRegisterCriteria буде використовуватися

#### delete

public boolean delete([I](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html) object)

Метод видаляє із бази даних всі дані пов'язанні з object, для пошука індефікатора використовується getRegisterCriteria

**Parameters:**

object - дані, які треба видалити

**Returns:**

true якщо видалення пройшло успішно і false якщо ні

#### deleteById

public boolean deleteById([I](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html) object)

Видаляє дані по їх індефікатору і всі інші дані пов'язані з ним. Не використовує getRegisterCriteria

**Parameters:**

object - дані для видалення

**Returns:**

true якщо видалення пройшло успішно і false якщо ні

#### getAllObjSatisfyFiltr

protected java.util.ArrayList<[I](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)> getAllObjSatisfyFiltr([Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html) filtr)

Метод витягує всі дані з таблиці, які задоволбняють фільтр

**Parameters:**

filtr - параметри фільтра

**Returns:**

повертає колекцію об'єктів, що задовольняють фільтр

#### register

public boolean register([I](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html) object)

Метод для регістрації об'єкта

**Parameters:**

object - об'єкт для збереження в базі даних

**Returns:**

повертає true якщо об'єкт успішно збережень у базу даних, false якщо ні

#### isRegisted

public [I](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html) isRegisted([I](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html) object)

Перевіряє чи об'єкт вже є в базі даних для цього використовує пошук getRegisterCriteria

**Parameters:**

object - об'єкт для перевірки

**Returns:**

повертає цей об'єкт із бази даних або null якщо таких даних немає

#### update

public boolean update([I](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html) object)

Метод обновлює дані в базі даних

**Parameters:**

object - дані, які необхідно оновити

**Returns:**

true якщо оновлення виконано успішно, false якщо ні

#### getAllObj

public java.util.ArrayList<[I](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)> getAllObj()

Метод повертає всі дані з таблиці

**Returns:**

повертає колекцію всіх даних із таблиці

#### deleteAllObj

public void deleteAllObj()

Видаляє всі дані з таблиці один за одгим

### Клас OperationsWithCards

### *Nested Class Summary*

|  |  |
| --- | --- |
| **Nested Classes** | |
| **Modifier and Type** | **Class and Description** |
| (package private) class | [**OperationsWithCards.Filtr**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.Filtr.html)  Клас, що описує поля таблиці, які будуть використовуватися для фільтру |

### *Field Summary*

|  |  |
| --- | --- |
| **Fields** | |
| **Modifier and Type** | **Field and Description** |
| private static [**OperationsWithCards**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.html) | [**oper**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.html#oper)  Поле призначене для зберігання силки на об'єкт цього класу |

### *Constructor Summary*

|  |  |
| --- | --- |
| **Constructors** | |
| **Modifier** | **Constructor and Description** |
| private | [**OperationsWithCards**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.html#OperationsWithCards--)()  Метод створюэ об'єкт цього класу |

### *Method Summary*

|  |  |
| --- | --- |
| **All Methods**[**Static Methods**](javascript:show(1);)[**Instance Methods**](javascript:show(2);)[**Concrete Methods**](javascript:show(8);) | |
| **Modifier and Type** | **Method and Description** |
| java.util.ArrayList<[**Card**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html)> | [**getAllCardsFiltr**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.html#getAllCardsFiltr-java.lang.String-hibernate.Gender-java.lang.Short-java.lang.Short-java.lang.Boolean-)(java.lang.String name,[**Gender**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Gender.html) gender, java.lang.Short age\_levelLow, java.lang.Short age\_levelHight, java.lang.Boolean isAgain)  Метод для отримання всіх даних з таблиці, що задовільняєть фільтр getFiltrCriteria |
| (package private) org.hibernate.Criteria | [**getAllObjCriteria**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.html#getAllObjCriteria-org.hibernate.Session-)(org.hibernate.Session session)  Абстрактний метод, який має реалізувати запит до бази даних |
| (package private) org.hibernate.Criteria | [**getFiltrCriteria**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.html#getFiltrCriteria-org.hibernate.Session-logic.OperationsWithCards.Filtr-)(org.hibernate.Session session,[**OperationsWithCards.Filtr**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.Filtr.html) filtr)  Абстрактний метод, який має реалізувати фільтр |
| static [**OperationsWithCards**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.html) | [**getOperationWithCard**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.html#getOperationWithCard--)()  Метод повертає об'єкт цього класу |
| (package private) org.hibernate.Criteria | [**getRegisterCriteria**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.html#getRegisterCriteria-org.hibernate.Session-hibernate.Card-)(org.hibernate.Session session,[**Card**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html) card)  Абстрактний метод, який реалізує критерії, за якими шукається дані в таблиці |
| (package private) boolean | [**isRegistedMethodNotNeed**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.html#isRegistedMethodNotNeed--)()  Абстрактний метод, який реалізує використання методу getRegisterCriteria |

### Methods inherited from class logic.[CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)

[delete](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#delete-I-), [deleteAllObj](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#deleteAllObj--), [deleteById](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#deleteById-I-), [getAllObj](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObj--), [getAllObjSatisfyFiltr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObjSatisfyFiltr-Filtr-),[isRegisted](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#isRegisted-I-), [register](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#register-I-), [update](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#update-I-)

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### *Field Detail*

#### oper

private static [OperationsWithCards](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.html) oper

Поле призначене для зберігання силки на об'єкт цього класу

### *Constructor Detail*

#### OperationsWithCards

private OperationsWithCards()

Метод створюэ об'єкт цього класу

### *Method Detail*

#### getOperationWithCard

public static [OperationsWithCards](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.html) getOperationWithCard()

Метод повертає об'єкт цього класу

**Returns:**

повертає об'єкт класу

#### getRegisterCriteria

* + - org.hibernate.Criteria getRegisterCriteria(org.hibernate.Session session,

[Card](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html) card)

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getRegisterCriteria-org.hibernate.Session-I-)

Абстрактний метод, який реалізує критерії, за якими шукається дані в таблиці

**Specified by:**

[getRegisterCriteria](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getRegisterCriteria-org.hibernate.Session-I-) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[Card](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html),[OperationsWithCards.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.Filtr.html)>

**Parameters:**

session - сесія з базою даних

card - об'єкт, який шукається в базі даних

**Returns:**

запит до бази даних для витягування всіх об'єктів, які задовільняють критерії

#### getFiltrCriteria

* + - org.hibernate.Criteria getFiltrCriteria(org.hibernate.Session session,

[OperationsWithCards.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.Filtr.html) filtr)

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getFiltrCriteria-org.hibernate.Session-Filtr-)

Абстрактний метод, який має реалізувати фільтр

**Specified by:**

[getFiltrCriteria](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getFiltrCriteria-org.hibernate.Session-Filtr-) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[Card](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html),[OperationsWithCards.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.Filtr.html)>

**Parameters:**

session - сесія з базою даних

filtr - параметри для фільтрації даних з бази даних

**Returns:**

запит до бази даних для витягування всіх об'єктів, які задовільняють фільтр

#### getAllObjCriteria

org.hibernate.Criteria getAllObjCriteria(org.hibernate.Session session)

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObjCriteria-org.hibernate.Session-)

Абстрактний метод, який має реалізувати запит до бази даних

**Specified by:**

[getAllObjCriteria](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObjCriteria-org.hibernate.Session-) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[Card](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html),[OperationsWithCards.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.Filtr.html)>

**Parameters:**

session - сесія з базою даних

**Returns:**

запит до бази даних для витягування всіх об'єктів класу "I"

#### getAllCardsFiltr

* + - public java.util.ArrayList<[Card](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html)> getAllCardsFiltr(java.lang.String name,

[Gender](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Gender.html) gender, java.lang.Short age\_levelLow,

java.lang.Short age\_levelHight,

java.lang.Boolean isAgain)

Метод для отримання всіх даних з таблиці, що задовільняєть фільтр getFiltrCriteria

**Parameters:**

name - задання ім'я хворого

gender - задання статі хворого

age\_levelLow - задання нижчого порогу віку

age\_levelHight - задання вищого порогу віку

isAgain - задання чи хворий не вперше

**Returns:**

колекцію карт

#### isRegistedMethodNotNeed

boolean isRegistedMethodNotNeed()

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#isRegistedMethodNotNeed--)

Абстрактний метод, який реалізує використання методу getRegisterCriteria

**Specified by:**

[isRegistedMethodNotNeed](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#isRegistedMethodNotNeed--) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[Card](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Card.html),[OperationsWithCards.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithCards.Filtr.html)>

**Returns:**

повертати true потрібно, коли використання метода getRegisterCriteria не потрібно (збільшує продуктивність), при повернені false метод getRegisterCriteria буде використовуватися

### Клас OperationsWithDiagnosis

### *Nested Class Summary*

|  |  |
| --- | --- |
| **Nested Classes** | |
| **Modifier and Type** | **Class and Description** |
| (package private) class | [**OperationsWithDiagnosis.Filtr**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.Filtr.html)  Клас, що описує поля таблиці, які будуть використовуватися для фільтру |

### *Field Summary*

|  |  |
| --- | --- |
| **Fields** | |
| **Modifier and Type** | **Field and Description** |
| private static [**OperationsWithDiagnosis**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.html) | [**oper**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.html#oper)  Поле призначене для зберігання силки на об'єкт цього класу |

### *Constructor Summary*

|  |  |
| --- | --- |
| **Constructors** | |
| **Modifier** | **Constructor and Description** |
| private | [**OperationsWithDiagnosis**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.html#OperationsWithDiagnosis--)()  Метод створюэ об'єкт цього класу |

### *Method Summary*

|  |  |
| --- | --- |
| **All Methods**[**Static Methods**](javascript:show(1);)[**Instance Methods**](javascript:show(2);)[**Concrete Methods**](javascript:show(8);) | |
| **Modifier and Type** | **Method and Description** |
| (package private) org.hibernate.Criteria | [**getAllObjCriteria**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.html#getAllObjCriteria-org.hibernate.Session-)(org.hibernate.Session session)  Абстрактний метод, який має реалізувати запит до бази даних |
| (package private) org.hibernate.Criteria | [**getFiltrCriteria**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.html#getFiltrCriteria-org.hibernate.Session-logic.OperationsWithDiagnosis.Filtr-)(org.hibernate.Session session,[**OperationsWithDiagnosis.Filtr**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.Filtr.html) filtr)  Абстрактний метод, який має реалізувати фільтр |
| static [**OperationsWithDiagnosis**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.html) | [**getOperationsWithDiagnosis**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.html#getOperationsWithDiagnosis--)()  Метод повертає об'єкт цього класу |
| (package private) org.hibernate.Criteria | [**getRegisterCriteria**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.html#getRegisterCriteria-org.hibernate.Session-hibernate.Diagnos-)(org.hibernate.Session session,[**Diagnos**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Diagnos.html) object)  Абстрактний метод, який реалізує критерії, за якими шукається дані в таблиці |
| (package private) boolean | [**isRegistedMethodNotNeed**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.html#isRegistedMethodNotNeed--)()  Абстрактний метод, який реалізує використання методу getRegisterCriteria |

### Methods inherited from class logic.[CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)

[delete](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#delete-I-), [deleteAllObj](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#deleteAllObj--), [deleteById](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#deleteById-I-), [getAllObj](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObj--), [getAllObjSatisfyFiltr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObjSatisfyFiltr-Filtr-),[isRegisted](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#isRegisted-I-), [register](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#register-I-), [update](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#update-I-)

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### *Field Detail*

#### oper

private static [OperationsWithDiagnosis](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.html) oper

Поле призначене для зберігання силки на об'єкт цього класу

### *Constructor Detail*

#### OperationsWithDiagnosis

private OperationsWithDiagnosis()

Метод створюэ об'єкт цього класу

### *Method Detail*

#### getOperationsWithDiagnosis

public static [OperationsWithDiagnosis](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.html) getOperationsWithDiagnosis()

Метод повертає об'єкт цього класу

**Returns:**

повертає об'єкт класу

#### getFiltrCriteria

* + - org.hibernate.Criteria getFiltrCriteria(org.hibernate.Session session,

[OperationsWithDiagnosis.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.Filtr.html) filtr)

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getFiltrCriteria-org.hibernate.Session-Filtr-)

Абстрактний метод, який має реалізувати фільтр

**Specified by:**

[getFiltrCriteria](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getFiltrCriteria-org.hibernate.Session-Filtr-) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[Diagnos](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Diagnos.html),[OperationsWithDiagnosis.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.Filtr.html)>

**Parameters:**

session - сесія з базою даних

filtr - параметри для фільтрації даних з бази даних

**Returns:**

запит до бази даних для витягування всіх об'єктів, які задовільняють фільтр

#### getAllObjCriteria

org.hibernate.Criteria getAllObjCriteria(org.hibernate.Session session)

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObjCriteria-org.hibernate.Session-)

Абстрактний метод, який має реалізувати запит до бази даних

**Specified by:**

[getAllObjCriteria](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObjCriteria-org.hibernate.Session-) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[Diagnos](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Diagnos.html),[OperationsWithDiagnosis.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.Filtr.html)>

**Parameters:**

session - сесія з базою даних

**Returns:**

запит до бази даних для витягування всіх об'єктів класу "I"

#### getRegisterCriteria

* + - org.hibernate.Criteria getRegisterCriteria(org.hibernate.Session session,

[Diagnos](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Diagnos.html) object)

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getRegisterCriteria-org.hibernate.Session-I-)

Абстрактний метод, який реалізує критерії, за якими шукається дані в таблиці

**Specified by:**

[getRegisterCriteria](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getRegisterCriteria-org.hibernate.Session-I-) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[Diagnos](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Diagnos.html),[OperationsWithDiagnosis.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.Filtr.html)>

**Parameters:**

session - сесія з базою даних

object - об'єкт, який шукається в базі даних

**Returns:**

запит до бази даних для витягування всіх об'єктів, які задовільняють критерії

#### isRegistedMethodNotNeed

boolean isRegistedMethodNotNeed()

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#isRegistedMethodNotNeed--)

Абстрактний метод, який реалізує використання методу getRegisterCriteria

**Specified by:**

[isRegistedMethodNotNeed](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#isRegistedMethodNotNeed--) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[Diagnos](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Diagnos.html),[OperationsWithDiagnosis.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithDiagnosis.Filtr.html)>

**Returns:**

повертати true потрібно, коли використання метода getRegisterCriteria не потрібно (збільшує продуктивність), при повернені false метод getRegisterCriteria буде використовуватися

### Клас OperationsWithUsers

### *Nested Class Summary*

|  |  |
| --- | --- |
| **Nested Classes** | |
| **Modifier and Type** | **Class and Description** |
| (package private) class | [**OperationsWithUsers.Filtr**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.Filtr.html)  Клас, що описує поля таблиці, які будуть використовуватися для фільтру |

### *Field Summary*

|  |  |
| --- | --- |
| **Fields** | |
| **Modifier and Type** | **Field and Description** |
| private static [**OperationsWithUsers**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.html) | [**operWithUsers**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.html#operWithUsers)  Поле призначене для зберігання силки на об'єкт цього класу |

### *Constructor Summary*

|  |  |
| --- | --- |
| **Constructors** | |
| **Modifier** | **Constructor and Description** |
| private | [**OperationsWithUsers**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.html#OperationsWithUsers--)()  Метод створюэ об'єкт цього класу |

### *Method Summary*

|  |  |
| --- | --- |
| **All Methods**[**Static Methods**](javascript:show(1);)[**Instance Methods**](javascript:show(2);)[**Concrete Methods**](javascript:show(8);) | |
| **Modifier and Type** | **Method and Description** |
| (package private) org.hibernate.Criteria | [**getAllObjCriteria**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.html#getAllObjCriteria-org.hibernate.Session-)(org.hibernate.Session session)  Абстрактний метод, який має реалізувати запит до бази даних |
| java.util.ArrayList<[**User**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\User.html)> | [**getAllObjSatisfyFiltr**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.html#getAllObjSatisfyFiltr-java.lang.String-java.lang.String-java.lang.Short-java.lang.Short-)(java.lang.String name, java.lang.String phone, java.lang.Short access\_levelLow, java.lang.Short access\_levelHight)  Метод для отримання всіх даних з таблиці, що задовільняєть фільтр getFiltrCriteria |
| (package private) org.hibernate.Criteria | [**getFiltrCriteria**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.html#getFiltrCriteria-org.hibernate.Session-logic.OperationsWithUsers.Filtr-)(org.hibernate.Session session,[**OperationsWithUsers.Filtr**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.Filtr.html) filtr)  Абстрактний метод, який має реалізувати фільтр |
| static [**OperationsWithUsers**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.html) | [**getOperationWithUsers**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.html#getOperationWithUsers--)()  Метод повертає об'єкт цього класу |
| (package private) org.hibernate.Criteria | [**getRegisterCriteria**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.html#getRegisterCriteria-org.hibernate.Session-hibernate.User-)(org.hibernate.Session session,[**User**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\User.html) user)  Абстрактний метод, який реалізує критерії, за якими шукається дані в таблиці |
| (package private) boolean | [**isRegistedMethodNotNeed**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.html#isRegistedMethodNotNeed--)()  Абстрактний метод, який реалізує використання методу getRegisterCriteria |

### Methods inherited from class logic.[CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)

[delete](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#delete-I-), [deleteAllObj](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#deleteAllObj--), [deleteById](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#deleteById-I-), [getAllObj](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObj--), [getAllObjSatisfyFiltr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObjSatisfyFiltr-Filtr-),[isRegisted](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#isRegisted-I-), [register](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#register-I-), [update](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#update-I-)

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### *Field Detail*

#### operWithUsers

private static [OperationsWithUsers](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.html) operWithUsers

Поле призначене для зберігання силки на об'єкт цього класу

### *Constructor Detail*

#### OperationsWithUsers

private OperationsWithUsers()

Метод створюэ об'єкт цього класу

### *Method Detail*

#### getOperationWithUsers

public static [OperationsWithUsers](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.html) getOperationWithUsers()

Метод повертає об'єкт цього класу

**Returns:**

повертає об'єкт класу

#### getRegisterCriteria

* + - org.hibernate.Criteria getRegisterCriteria(org.hibernate.Session session,

[User](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\User.html) user)

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getRegisterCriteria-org.hibernate.Session-I-)

Абстрактний метод, який реалізує критерії, за якими шукається дані в таблиці

**Specified by:**

[getRegisterCriteria](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getRegisterCriteria-org.hibernate.Session-I-) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[User](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\User.html),[OperationsWithUsers.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.Filtr.html)>

**Parameters:**

session - сесія з базою даних

user - об'єкт, який шукається в базі даних

**Returns:**

запит до бази даних для витягування всіх об'єктів, які задовільняють критерії

#### getFiltrCriteria

* + - org.hibernate.Criteria getFiltrCriteria(org.hibernate.Session session,

[OperationsWithUsers.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.Filtr.html) filtr)

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getFiltrCriteria-org.hibernate.Session-Filtr-)

Абстрактний метод, який має реалізувати фільтр

**Specified by:**

[getFiltrCriteria](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getFiltrCriteria-org.hibernate.Session-Filtr-) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[User](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\User.html),[OperationsWithUsers.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.Filtr.html)>

**Parameters:**

session - сесія з базою даних

filtr - параметри для фільтрації даних з бази даних

**Returns:**

запит до бази даних для витягування всіх об'єктів, які задовільняють фільтр

#### getAllObjCriteria

org.hibernate.Criteria getAllObjCriteria(org.hibernate.Session session)

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObjCriteria-org.hibernate.Session-)

Абстрактний метод, який має реалізувати запит до бази даних

**Specified by:**

[getAllObjCriteria](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObjCriteria-org.hibernate.Session-) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[User](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\User.html),[OperationsWithUsers.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.Filtr.html)>

**Parameters:**

session - сесія з базою даних

**Returns:**

запит до бази даних для витягування всіх об'єктів класу "I"

#### getAllObjSatisfyFiltr

* + - public java.util.ArrayList<[User](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\User.html)> getAllObjSatisfyFiltr(java.lang.String name,
    - java.lang.String phone,
    - java.lang.Short access\_levelLow,

java.lang.Short access\_levelHight)

Метод для отримання всіх даних з таблиці, що задовільняєть фільтр getFiltrCriteria

**Parameters:**

name - задання ім'я користувача

phone - задання телефону користувача

access\_levelLow - задання нижчого порогу доступу

access\_levelHight - задання вищого порогу доступу

**Returns:**

колекцію користувачів

#### isRegistedMethodNotNeed

boolean isRegistedMethodNotNeed()

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#isRegistedMethodNotNeed--)

Абстрактний метод, який реалізує використання методу getRegisterCriteria

**Specified by:**

[isRegistedMethodNotNeed](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#isRegistedMethodNotNeed--) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[User](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\User.html),[OperationsWithUsers.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationsWithUsers.Filtr.html)>

**Returns:**

повертати true потрібно, коли використання метода getRegisterCriteria не потрібно (збільшує продуктивність), при повернені false метод getRegisterCriteria буде використовуватися

### Клас OperationsWithNotes

### *Nested Class Summary*

|  |  |
| --- | --- |
| **Nested Classes** | |
| **Modifier and Type** | **Class and Description** |
| (package private) class | [**OperationWithNotes.Filtr**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.Filtr.html)  Клас, що описує поля таблиці, які будуть використовуватися для фільтру |

### *Field Summary*

|  |  |
| --- | --- |
| **Fields** | |
| **Modifier and Type** | **Field and Description** |
| private static [**OperationWithNotes**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.html) | [**oper**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.html#oper)  Поле призначене для зберігання силки на об'єкт цього класу |

### *Constructor Summary*

|  |  |
| --- | --- |
| **Constructors** | |
| **Modifier** | **Constructor and Description** |
| private | [**OperationWithNotes**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.html#OperationWithNotes--)()  Метод створюэ об'єкт цього класу |

### *Method Summary*

|  |  |
| --- | --- |
| **All Methods**[**Static Methods**](javascript:show(1);)[**Instance Methods**](javascript:show(2);)[**Concrete Methods**](javascript:show(8);) | |
| **Modifier and Type** | **Method and Description** |
| (package private) org.hibernate.Criteria | [**getAllObjCriteria**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.html#getAllObjCriteria-org.hibernate.Session-)(org.hibernate.Session session)  Абстрактний метод, який має реалізувати запит до бази даних |
| java.util.ArrayList<[**Note**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Note.html)> | [**getAllObjSatisfyFiltr**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.html#getAllObjSatisfyFiltr-java.lang.Boolean-java.sql.Date-java.sql.Date-)(java.lang.Boolean hide, java.sql.Date date\_low, java.sql.Date date\_hight)  Метод для отримання всіх даних з таблиці, що задовільняєть фільтр getFiltrCriteria |
| (package private) org.hibernate.Criteria | [**getFiltrCriteria**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.html#getFiltrCriteria-org.hibernate.Session-logic.OperationWithNotes.Filtr-)(org.hibernate.Session session,[**OperationWithNotes.Filtr**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.Filtr.html) filtr)  Абстрактний метод, який має реалізувати фільтр |
| static [**OperationWithNotes**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.html) | [**getOperationWithNotes**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.html#getOperationWithNotes--)()  Метод повертає об'єкт цього класу |
| (package private) org.hibernate.Criteria | [**getRegisterCriteria**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.html#getRegisterCriteria-org.hibernate.Session-hibernate.Note-)(org.hibernate.Session session,[**Note**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Note.html) object)  Абстрактний метод, який реалізує критерії, за якими шукається дані в таблиці |
| (package private) boolean | [**isRegistedMethodNotNeed**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.html#isRegistedMethodNotNeed--)()  Абстрактний метод, який реалізує використання методу getRegisterCriteria |

### Methods inherited from class logic.[CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)

[delete](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#delete-I-), [deleteAllObj](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#deleteAllObj--), [deleteById](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#deleteById-I-), [getAllObj](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObj--), [getAllObjSatisfyFiltr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObjSatisfyFiltr-Filtr-),[isRegisted](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#isRegisted-I-), [register](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#register-I-), [update](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#update-I-)

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### *Field Detail*

#### oper

private static [OperationWithNotes](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.html) oper

Поле призначене для зберігання силки на об'єкт цього класу

### *Constructor Detail*

#### OperationWithNotes

private OperationWithNotes()

Метод створюэ об'єкт цього класу

### *Method Detail*

#### getOperationWithNotes

public static [OperationWithNotes](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.html) getOperationWithNotes()

Метод повертає об'єкт цього класу

**Returns:**

повертає об'єкт класу

#### getRegisterCriteria

* + - org.hibernate.Criteria getRegisterCriteria(org.hibernate.Session session,

[Note](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Note.html) object)

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getRegisterCriteria-org.hibernate.Session-I-)

Абстрактний метод, який реалізує критерії, за якими шукається дані в таблиці

**Specified by:**

[getRegisterCriteria](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getRegisterCriteria-org.hibernate.Session-I-) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[Note](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Note.html),[OperationWithNotes.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.Filtr.html)>

**Parameters:**

session - сесія з базою даних

object - об'єкт, який шукається в базі даних

**Returns:**

запит до бази даних для витягування всіх об'єктів, які задовільняють критерії

#### getFiltrCriteria

* + - org.hibernate.Criteria getFiltrCriteria(org.hibernate.Session session,

[OperationWithNotes.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.Filtr.html) filtr)

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getFiltrCriteria-org.hibernate.Session-Filtr-)

Абстрактний метод, який має реалізувати фільтр

**Specified by:**

[getFiltrCriteria](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getFiltrCriteria-org.hibernate.Session-Filtr-) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[Note](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Note.html),[OperationWithNotes.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.Filtr.html)>

**Parameters:**

session - сесія з базою даних

filtr - параметри для фільтрації даних з бази даних

**Returns:**

запит до бази даних для витягування всіх об'єктів, які задовільняють фільтр

#### getAllObjCriteria

org.hibernate.Criteria getAllObjCriteria(org.hibernate.Session session)

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObjCriteria-org.hibernate.Session-)

Абстрактний метод, який має реалізувати запит до бази даних

**Specified by:**

[getAllObjCriteria](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObjCriteria-org.hibernate.Session-) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[Note](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Note.html),[OperationWithNotes.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.Filtr.html)>

**Parameters:**

session - сесія з базою даних

**Returns:**

запит до бази даних для витягування всіх об'єктів класу "I"

#### isRegistedMethodNotNeed

boolean isRegistedMethodNotNeed()

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#isRegistedMethodNotNeed--)

Абстрактний метод, який реалізує використання методу getRegisterCriteria

**Specified by:**

[isRegistedMethodNotNeed](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#isRegistedMethodNotNeed--) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[Note](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Note.html),[OperationWithNotes.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithNotes.Filtr.html)>

**Returns:**

повертати true потрібно, коли використання метода getRegisterCriteria не потрібно (збільшує продуктивність), при повернені false метод getRegisterCriteria буде використовуватися

#### getAllObjSatisfyFiltr

* + - public java.util.ArrayList<[Note](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Note.html)> getAllObjSatisfyFiltr(java.lang.Boolean hide,
    - java.sql.Date date\_low,

java.sql.Date date\_hight)

Метод для отримання всіх даних з таблиці, що задовільняєть фільтр getFiltrCriteria

**Parameters:**

hide - задання стану нотатку

date\_low - задання нижчого порогу часу

date\_hight - задання вищого порогу часу

**Returns:**

колекцію нотатків

### Клас OperationsWithSession

### *Nested Class Summary*

|  |  |
| --- | --- |
| **Nested Classes** | |
| **Modifier and Type** | **Class and Description** |
| (package private) class | [**OperationWithSessions.Filtr**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.Filtr.html)  Клас, що описує поля таблиці, які будуть використовуватися для фільтру |

### *Field Summary*

|  |  |
| --- | --- |
| **Fields** | |
| **Modifier and Type** | **Field and Description** |
| private static [**OperationWithSessions**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.html) | [**oper**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.html#oper)  Поле призначене для зберігання силки на об'єкт цього класу |

### *Constructor Summary*

|  |  |
| --- | --- |
| **Constructors** | |
| **Modifier** | **Constructor and Description** |
| private | [**OperationWithSessions**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.html#OperationWithSessions--)()  Метод створюэ об'єкт цього класу |

|  |  |
| --- | --- |
| **All Methods**[**Static Methods**](javascript:show(1);)[**Instance Methods**](javascript:show(2);)[**Concrete Methods**](javascript:show(8);) | |
| **Modifier and Type** | **Method and Description** |
| (package private) org.hibernate.Criteria | [**getAllObjCriteria**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.html#getAllObjCriteria-org.hibernate.Session-)(org.hibernate.Session session)  Абстрактний метод, який має реалізувати запит до бази даних |
| java.util.ArrayList<[**Session**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Session.html)> | [**getAllObjSatisfyFiltr**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.html#getAllObjSatisfyFiltr-java.lang.Boolean-)(java.lang.Boolean result)  Метод для отримання всіх даних з таблиці, що задовільняєть фільтр getFiltrCriteria |
| (package private) org.hibernate.Criteria | [**getFiltrCriteria**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.html#getFiltrCriteria-org.hibernate.Session-logic.OperationWithSessions.Filtr-)(org.hibernate.Session session,[**OperationWithSessions.Filtr**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.Filtr.html) filtr)  Абстрактний метод, який має реалізувати фільтр |
| static [**OperationWithSessions**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.html) | [**getOperationWithSessions**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.html#getOperationWithSessions--)()  Метод повертає об'єкт цього класу |
| (package private) org.hibernate.Criteria | [**getRegisterCriteria**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.html#getRegisterCriteria-org.hibernate.Session-hibernate.Session-)(org.hibernate.Session session,  [**Session**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Session.html) object)  Абстрактний метод, який реалізує критерії, за якими шукається  дані в таблиці |
| (package private) boolean | [**isRegistedMethodNotNeed**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.html#isRegistedMethodNotNeed--)()  Абстрактний метод, який реалізує використання методу  getRegisterCriteria |

### *Method Summary*

### Methods inherited from class logic.[CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)

[delete](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#delete-I-), [deleteAllObj](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#deleteAllObj--), [deleteById](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#deleteById-I-), [getAllObj](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObj--), [getAllObjSatisfyFiltr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObjSatisfyFiltr-Filtr-),[isRegisted](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#isRegisted-I-), [register](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#register-I-), [update](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#update-I-)

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### *Field Detail*

#### oper

private static [OperationWithSessions](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.html) oper

Поле призначене для зберігання силки на об'єкт цього класу

### *Constructor Detail*

#### OperationWithSessions

private OperationWithSessions()

Метод створюэ об'єкт цього класу

### *Method Detail*

#### getOperationWithSessions

public static [OperationWithSessions](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.html) getOperationWithSessions()

Метод повертає об'єкт цього класу

**Returns:**

повертає об'єкт класу

#### getFiltrCriteria

* + - org.hibernate.Criteria getFiltrCriteria(org.hibernate.Session session,

[OperationWithSessions.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.Filtr.html) filtr)

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getFiltrCriteria-org.hibernate.Session-Filtr-)

Абстрактний метод, який має реалізувати фільтр

**Specified by:**

[getFiltrCriteria](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getFiltrCriteria-org.hibernate.Session-Filtr-) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[Session](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Session.html),[OperationWithSessions.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.Filtr.html)>

**Parameters:**

session - сесія з базою даних

filtr - параметри для фільтрації даних з бази даних

**Returns:**

запит до бази даних для витягування всіх об'єктів, які задовільняють фільтр

#### getAllObjCriteria

org.hibernate.Criteria getAllObjCriteria(org.hibernate.Session session)

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObjCriteria-org.hibernate.Session-)

Абстрактний метод, який має реалізувати запит до бази даних

**Specified by:**

[getAllObjCriteria](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getAllObjCriteria-org.hibernate.Session-) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[Session](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Session.html),[OperationWithSessions.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.Filtr.html)>

**Parameters:**

session - сесія з базою даних

**Returns:**

запит до бази даних для витягування всіх об'єктів класу "I"

#### getRegisterCriteria

* + - org.hibernate.Criteria getRegisterCriteria(org.hibernate.Session session,

[Session](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Session.html) object)

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getRegisterCriteria-org.hibernate.Session-I-)

Абстрактний метод, який реалізує критерії, за якими шукається дані в таблиці

**Specified by:**

[getRegisterCriteria](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#getRegisterCriteria-org.hibernate.Session-I-) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[Session](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Session.html),[OperationWithSessions.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.Filtr.html)>

**Parameters:**

session - сесія з базою даних

object - об'єкт, який шукається в базі даних

**Returns:**

запит до бази даних для витягування всіх об'єктів, які задовільняють критерії

#### isRegistedMethodNotNeed

boolean isRegistedMethodNotNeed()

**Description copied from class:**[**CommonOperations**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#isRegistedMethodNotNeed--)

Абстрактний метод, який реалізує використання методу getRegisterCriteria

**Specified by:**

[isRegistedMethodNotNeed](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html#isRegistedMethodNotNeed--) in class [CommonOperations](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\CommonOperations.html)<[Session](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Session.html),[OperationWithSessions.Filtr](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\logic\OperationWithSessions.Filtr.html)>

**Returns:**

повертати true потрібно, коли використання метода getRegisterCriteria не потрібно (збільшує продуктивність), при повернені false метод getRegisterCriteria буде використовуватися

#### getAllObjSatisfyFiltr

public java.util.ArrayList<[Session](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\hibernate\Session.html)> getAllObjSatisfyFiltr(java.lang.Boolean result)

Метод для отримання всіх даних з таблиці, що задовільняєть фільтрgetFiltrCriteria

**Parameters:**

result - задання результату лікування

**Returns:**

колекцію сесій

## Класи сервелетів та їх специфікація

Ці класи призначення для створення зв’язку між програмою та графічним інтерфейсом програми. Класи надають той необхідний функціонал актору, який треба для виконання всіх операцій, які передбачені для нього. Діаграму класів дивись на рис. 3.1.

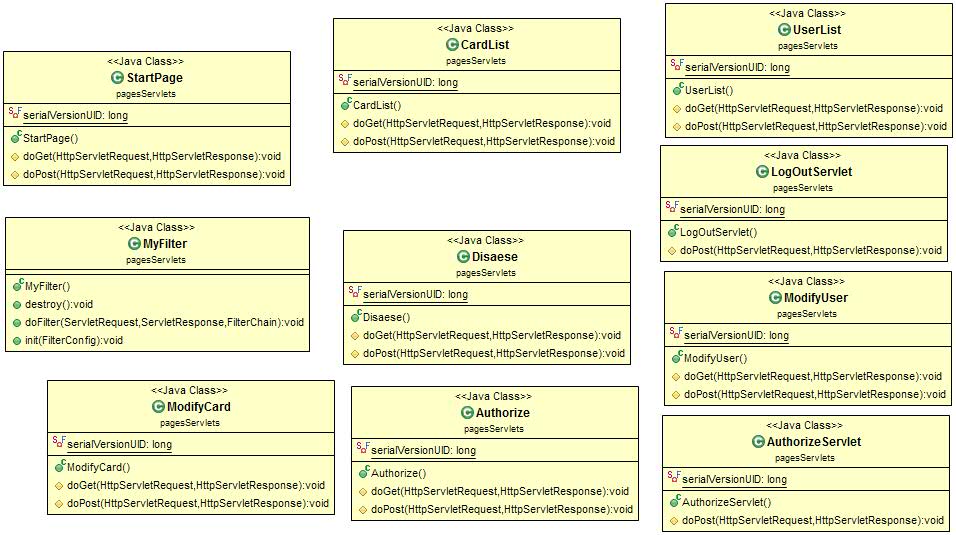


Рис. 3.3 Діаграма сервелетів

## 3.3.1 Класс AuthorizeServlet

### *Method Detail*

#### doPost

* + protected void doPost(javax.servlet.http.HttpServletRequest request,

javax.servlet.http.HttpServletResponse response)

throws javax.servlet.ServletException,

java.io.IOException

Метод приймає введені дані на формі користувачем і викликає метод AuthorizeUser.authorizeUser для йогот авторизації

**Overrides:**

doPost in class javax.servlet.http.HttpServlet

**Parameters:**

request - запит, який приходить від користувача

response - відповідь, яка надсилається користувачу

**Throws:**

javax.servlet.ServletException

java.io.IOException

### Клас LogOutServelet

### *Method Detail*

#### doPost

* + protected void doPost(javax.servlet.http.HttpServletRequest request,

javax.servlet.http.HttpServletResponse response)

throws javax.servlet.ServletException,

java.io.IOException

Метод виконує операцію виходу користувача і викликає метод AuthorizeUser.setUserLogOut для йогот виходу із системи

**Overrides:**

doPost in class javax.servlet.http.HttpServlet

**Parameters:**

request - запит, який приходить від користувача

response - відповідь, яка надсилається користувачу

**Throws:**

javax.servlet.ServletException

java.io.IOException

### Клас MyFilter

### *Method Detail*

#### destroy

public void destroy()

Метод викликається при знищені об'єкта фільтра

**Specified by:**

destroy in interface javax.servlet.Filter

#### doFilter

* + public void doFilter(javax.servlet.ServletRequest request,

javax.servlet.ServletResponse response,

javax.servlet.FilterChain chain)

throws java.io.IOException,

javax.servlet.ServletException

Метод викликається при будь-яких запитах і відповідях клієнту. Метод перевіряє чи користувач авторизован, якщо так то передати далі запит\відповідь, якщо ні перенаправити користувача на сторіну авторизації

**Specified by:**

doFilter in interface javax.servlet.Filter

**Parameters:**

request - запит клієнта

response - відповідь клієнта

chain - ланцюг фільтрів

**Throws:**

java.io.IOException

javax.servlet.ServletException

#### init

* + public void init(javax.servlet.FilterConfig fConfig)

throws javax.servlet.ServletException

Метод викликається при створені об'єкта фільтра

**Specified by:**

init in interface javax.servlet.Filter

**Parameters:**

fConfig - налаштування для фільтра

**Throws:**

javax.servlet.ServletException

### Клас Authorize

|  |  |
| --- | --- |
| **All Methods**[**Instance Methods**](javascript:show(2);)[**Concrete Methods**](javascript:show(8);) | |
| **Modifier and Type** | **Method and Description** |
| protected void | [**doGet**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\pagesServlets\Authorize.html#doGet-javax.servlet.http.HttpServletRequest-javax.servlet.http.HttpServletResponse-)(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)  Обробляє запит на вход до системи |

### Клас CardList

|  |  |
| --- | --- |
| **All Methods**[**Instance Methods**](javascript:show(2);)[**Concrete Methods**](javascript:show(8);) | |
| **Modifier and Type** | **Method and Description** |
| protected void | [**doGet**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\pagesServlets\CardList.html#doGet-javax.servlet.http.HttpServletRequest-javax.servlet.http.HttpServletResponse-)(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)  Обробляє запит на виведення сторінки з інформацією з таблиці cards |

### Клас Disease

|  |  |
| --- | --- |
| **All Methods**[**Instance Methods**](javascript:show(2);)[**Concrete Methods**](javascript:show(8);) | |
| **Modifier and Type** | **Method and Description** |
| protected void | [**doGet**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\pagesServlets\Disaese.html#doGet-javax.servlet.http.HttpServletRequest-javax.servlet.http.HttpServletResponse-)(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)  Обробляє запит на виведення сторінки з інформацією з таблиці disaeses |

### Клас ModifyCard

|  |  |
| --- | --- |
| **All Methods**[**Instance Methods**](javascript:show(2);)[**Concrete Methods**](javascript:show(8);) | |
| **Modifier and Type** | **Method and Description** |
| protected void | [**doGet**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\pagesServlets\ModifyCard.html#doGet-javax.servlet.http.HttpServletRequest-javax.servlet.http.HttpServletResponse-)(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)  Обробляє запит на виведення сторінки з інформацією з таблиці cards |

### Клас ModifyUser

### *Method Summary*

|  |  |
| --- | --- |
| **All Methods**[**Instance Methods**](javascript:show(2);)[**Concrete Methods**](javascript:show(8);) | |
| **Modifier and Type** | **Method and Description** |
| protected void | [**doGet**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\pagesServlets\ModifyUser.html#doGet-javax.servlet.http.HttpServletRequest-javax.servlet.http.HttpServletResponse-)(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)  Обробляє запит на виведення сторінки з інформацією з таблиці users |

### Клас StartPage

|  |  |
| --- | --- |
| **All Methods**[**Instance Methods**](javascript:show(2);)[**Concrete Methods**](javascript:show(8);) | |
| **Modifier and Type** | **Method and Description** |
| protected void | [**doGet**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\pagesServlets\StartPage.html#doGet-javax.servlet.http.HttpServletRequest-javax.servlet.http.HttpServletResponse-)(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)  Обробляє запит на виведення головної сторінки |

### Клас UserList

|  |  |
| --- | --- |
| **All Methods**[**Instance Methods**](javascript:show(2);)[**Concrete Methods**](javascript:show(8);) | |
| **Modifier and Type** | **Method and Description** |
| protected void | [**doGet**](file:///C:\Users\Chupakabra\Desktop\%D1%96%D0%B0\pagesServlets\UserList.html#doGet-javax.servlet.http.HttpServletRequest-javax.servlet.http.HttpServletResponse-)(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)  Обробляє запит на виведення сторінки з інформацією з таблиці users |

# РОЗДІЛ 4

# ІЛЮСТРАЦІЯ РОБОТИ ПРОГРАМИ

Для демонстрації роботи в цьому розділі буде наведені графічні сценарії роботи проекту.



## Ілюстрація сценарію «Входу в систему»

### Система надає форму для входу дивись Рис. 4.1

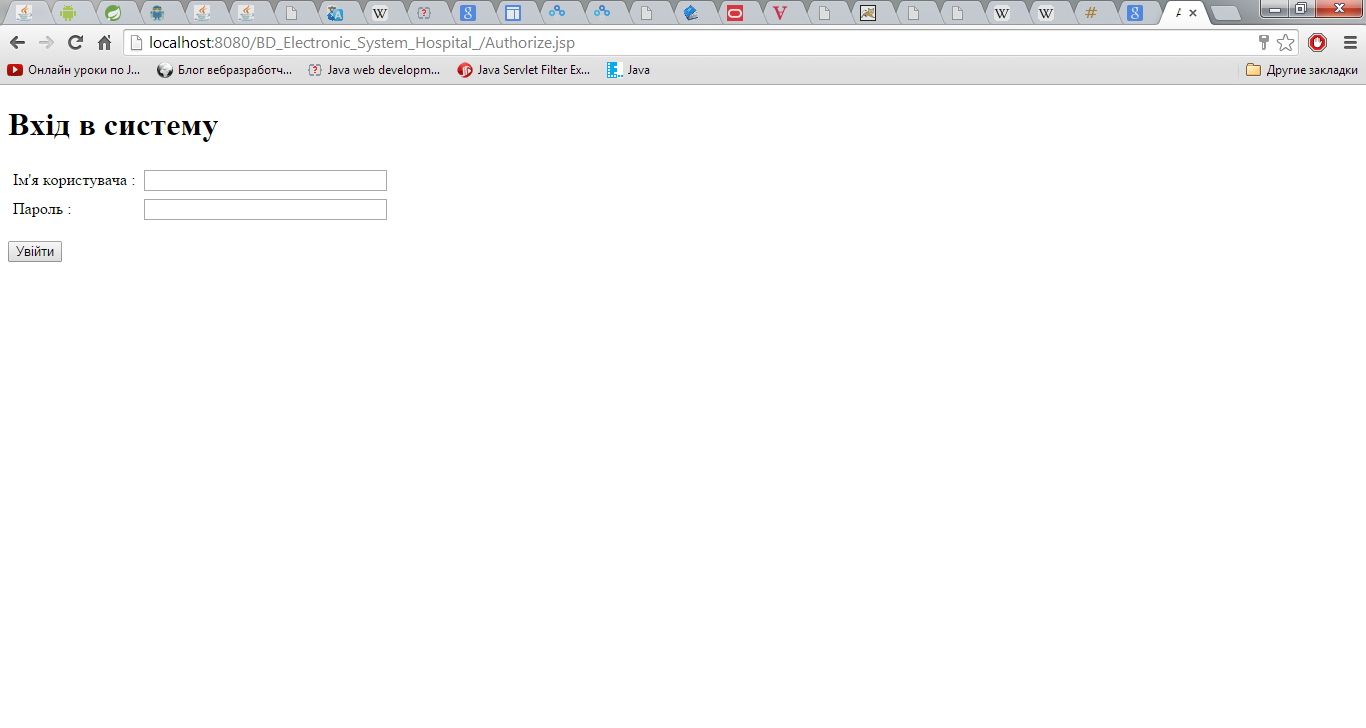


Рис. 4.1 Сторінка входу в систему

### Користувач заповнює поля дивись Рис. 4.2

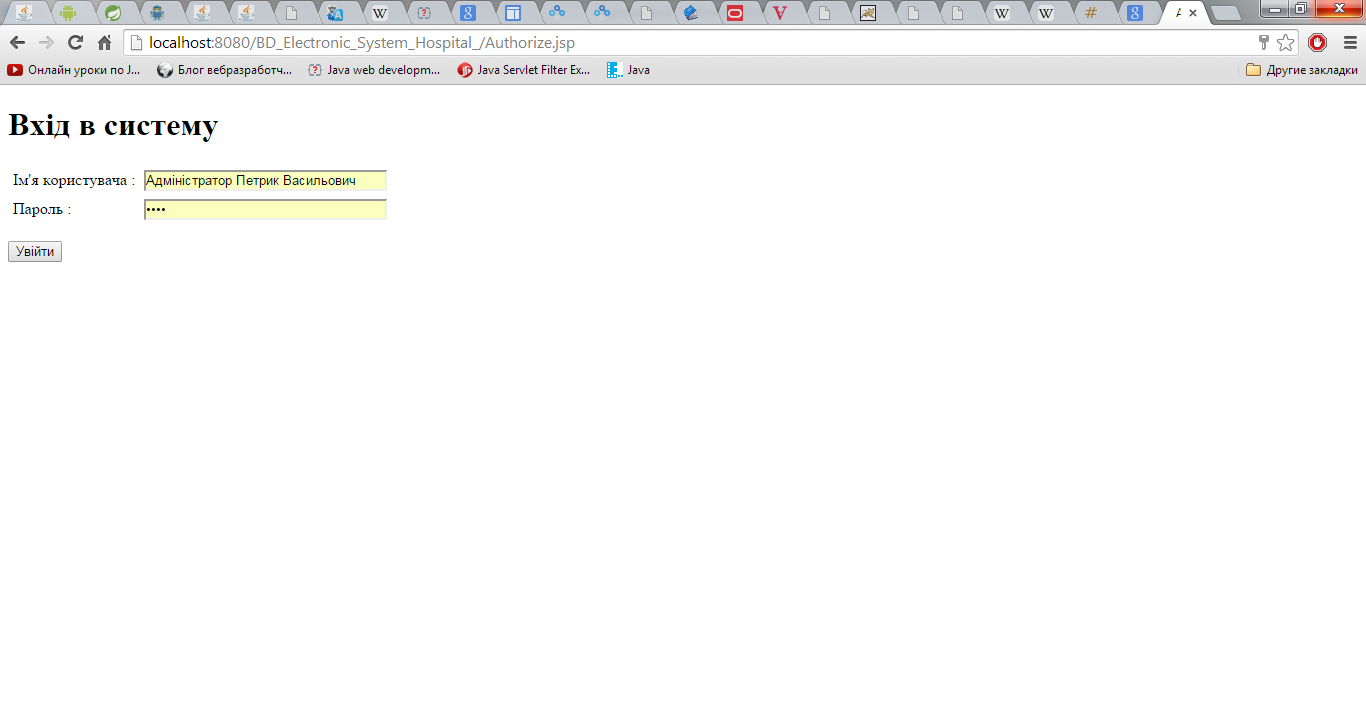


Рис. 4.2 Сторінка входу в систему з заповненими полями

### Користувача натискає кнопку «Увійти». При успішній авторизації користувач перейде на сторінку Рис. 4.3. При помилці авторизації користувач перейде на сторінку Рис. 4.4.

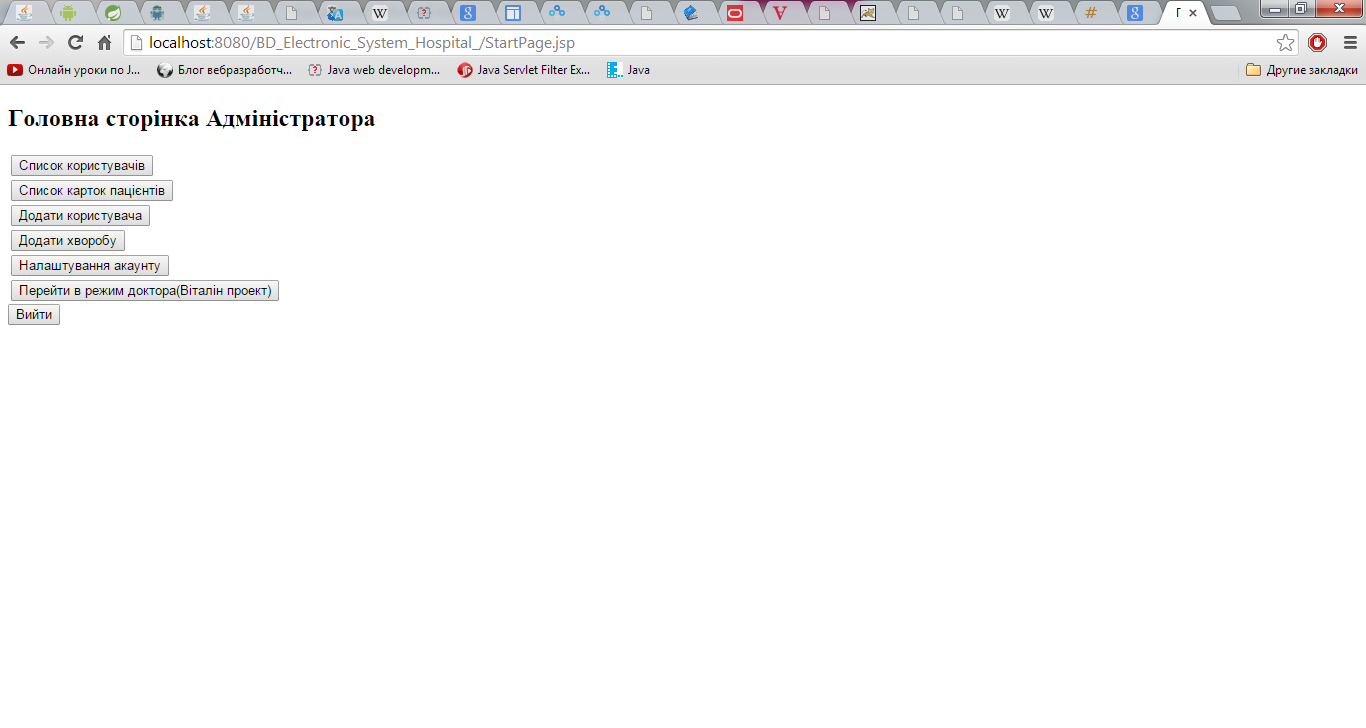


Рис. 4.3 Головна сторінка

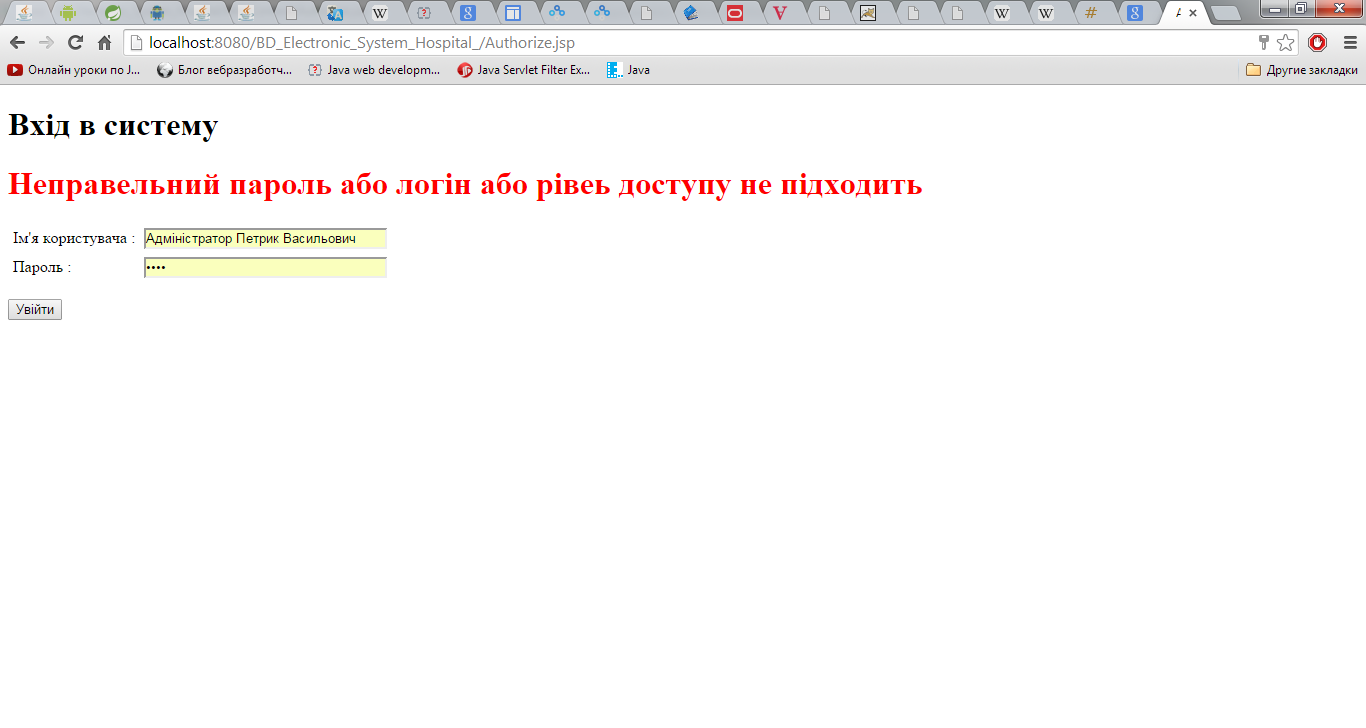


Рис. 4.4 Сторінка з помилкою входу

## Ілюстрація сценарію «Перегляд користувачів»

### Користувачу для переходу на сторінку «Перегляду користувачів» треба натиснути кнопку «Список користувачів» дивися Рис. 4.3

### Система пере направить користувача на сторінку Рис. 4.5

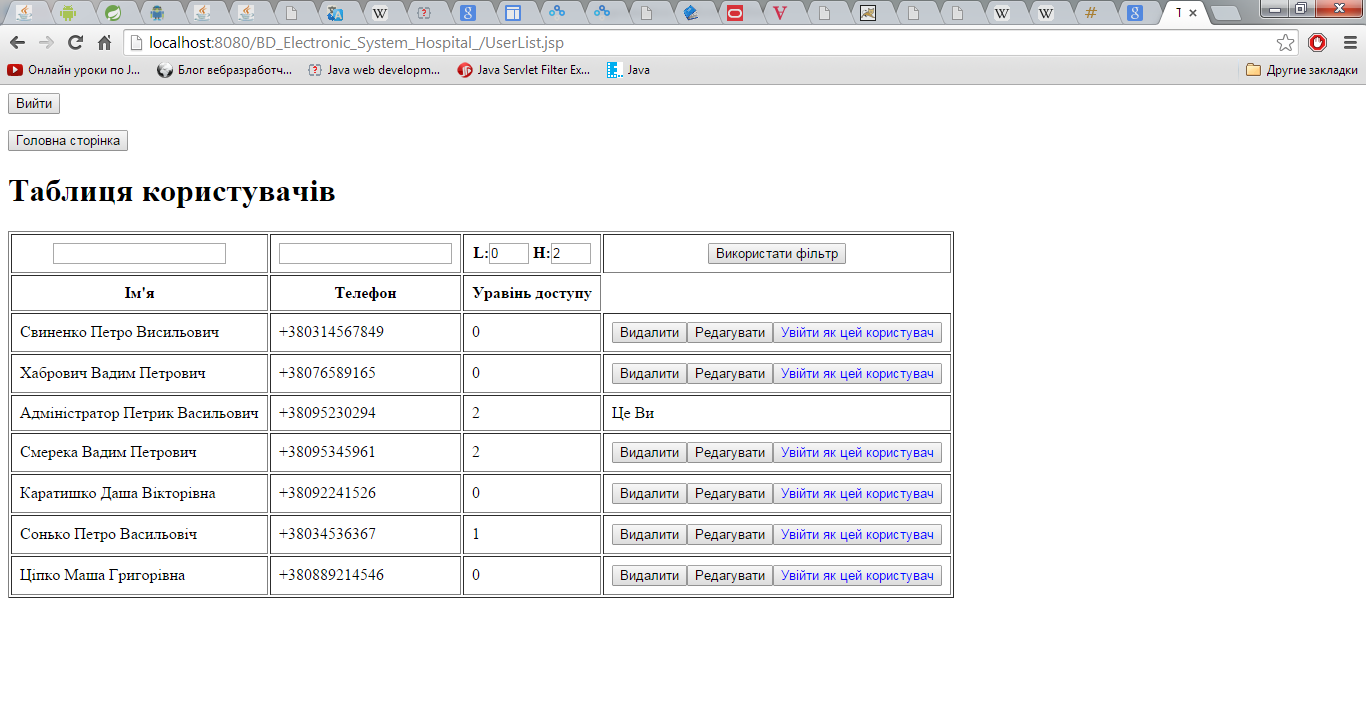


Рис. 4.5 Таблиця користувачів

## Ілюстрація сценарію «Перегляд карток»

### Користувачу для переходу на сторінку «Перегляд карток» треба натиснути кнопку «Список карток пацієнтів» дивися Рис. 4.3

### Система пере направить користувача на сторінку Рис. 4.6

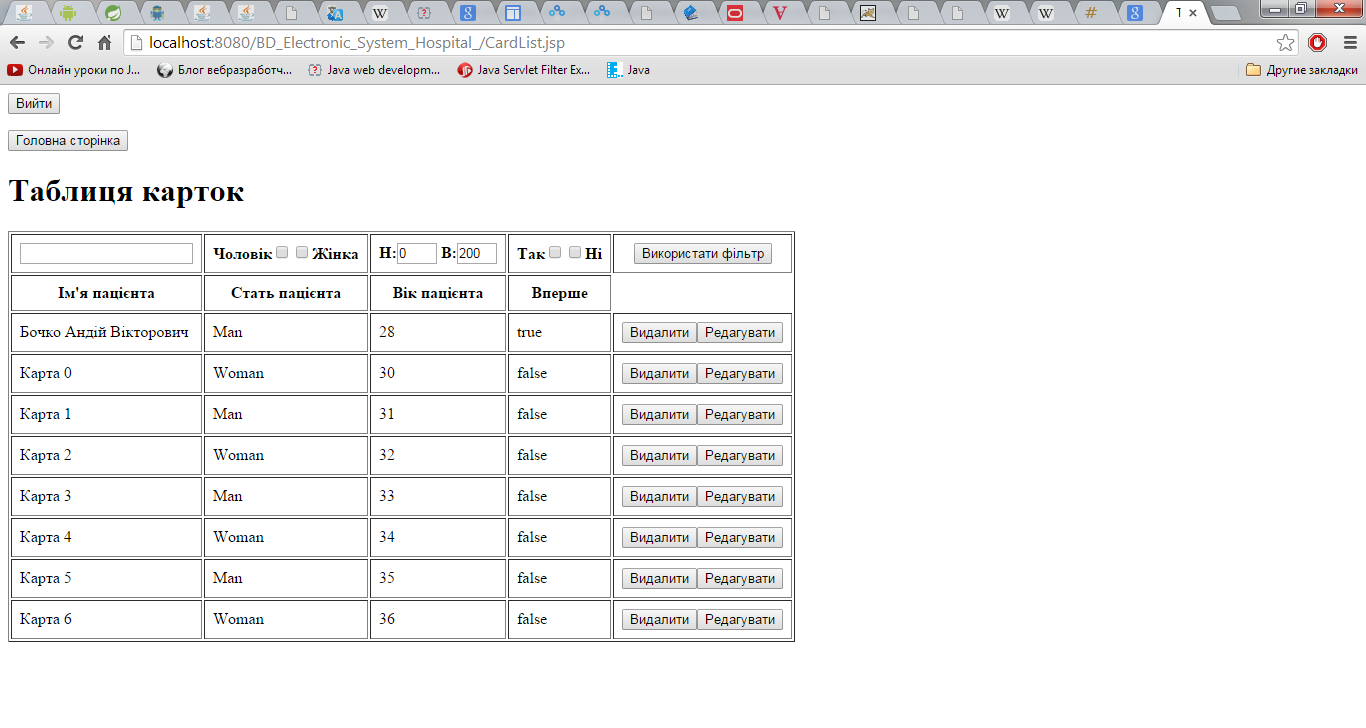


Рис. 4.6 Таблиця карток

## Ілюстрація сценарію «Створення нового користувача»

### Для створення аканта нового користувача необхідно натиснути на кнопку «Додати користувача» на сторінці Рис. 4.3.

### Система пере направить користувача на сторінку Рис. 4.7.

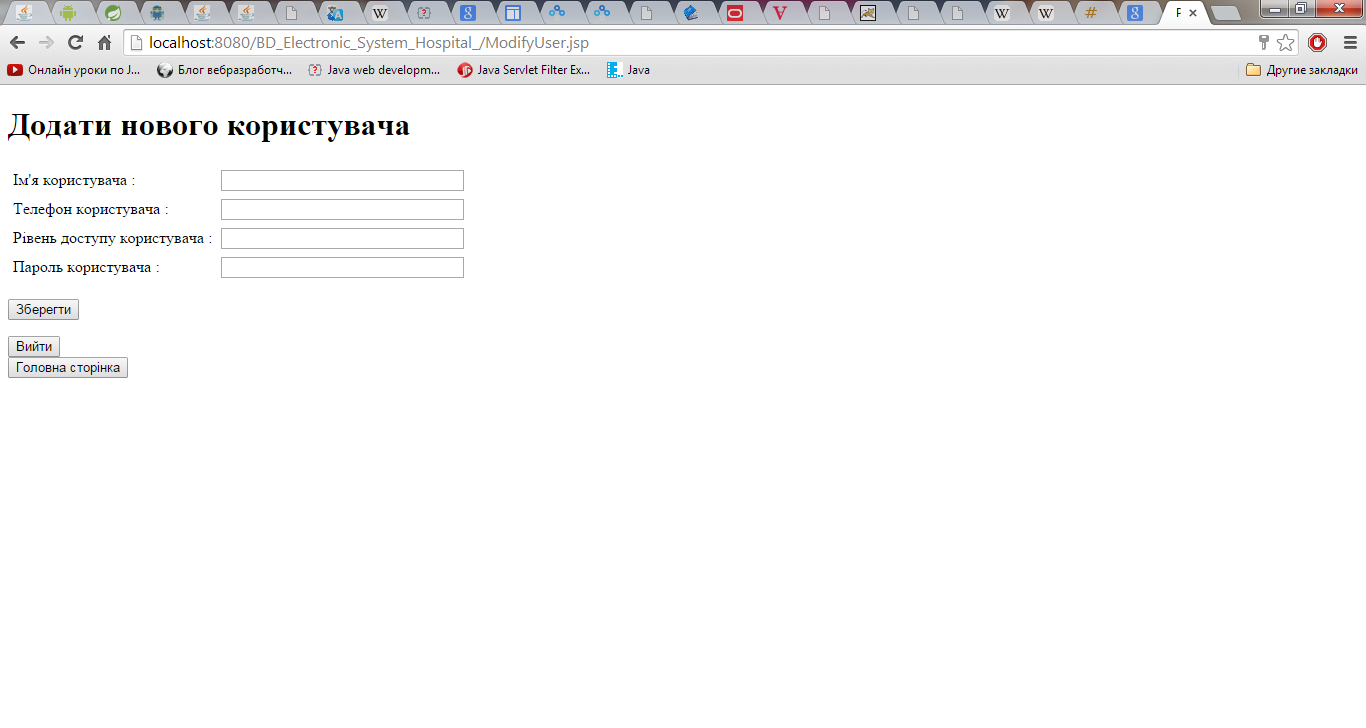


Рис. 4.7 Сторінка створення користувача

### Адміністратор заповнює поля форми дивись Рис. 4.8.

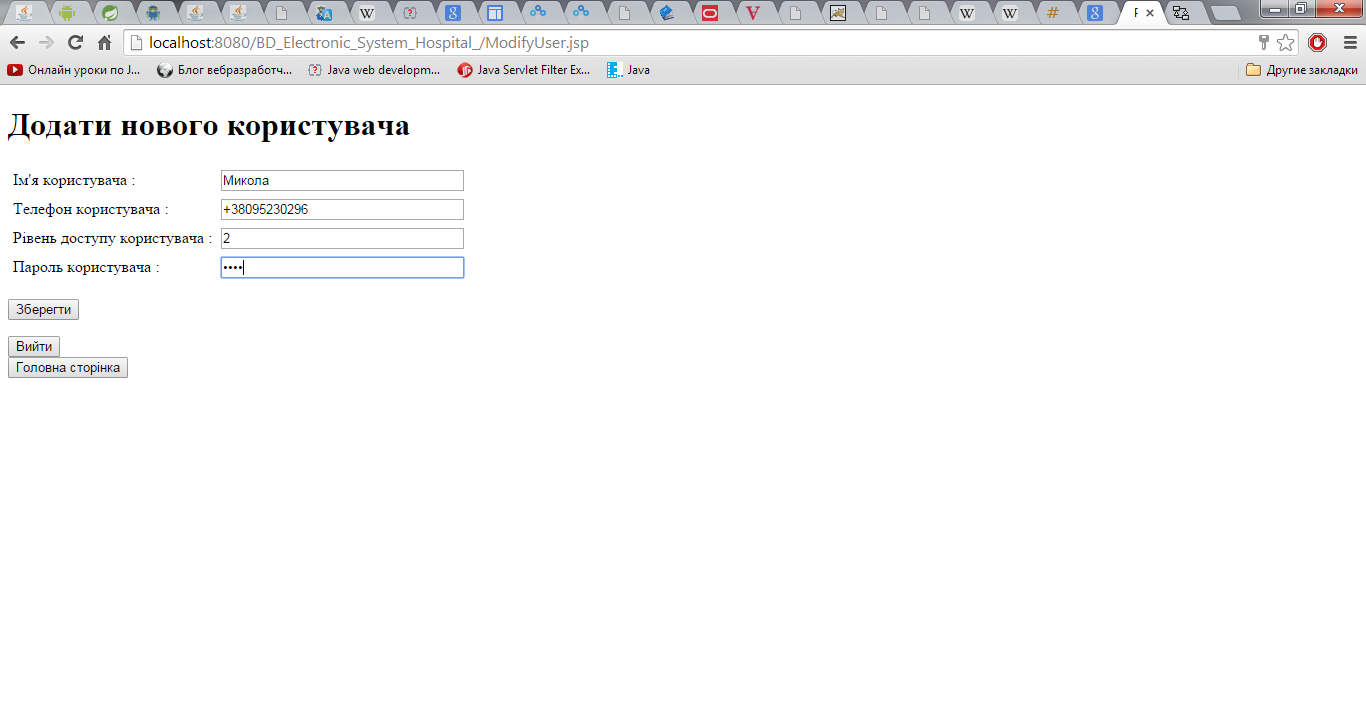


Рис. 4.8 Сторінка для внесення нового користувача

### Для збереження нового користувача адміністратор повинен натиснути кнопку «Зберегти». Система після успішного виконання операції пере направить адміністратора на сторінку Рис. 4.9. При неможливості виконати операцію система пере направить адміністратора на сторінку Рис. 4.9 із написом «Помилка» замість «Успішно»

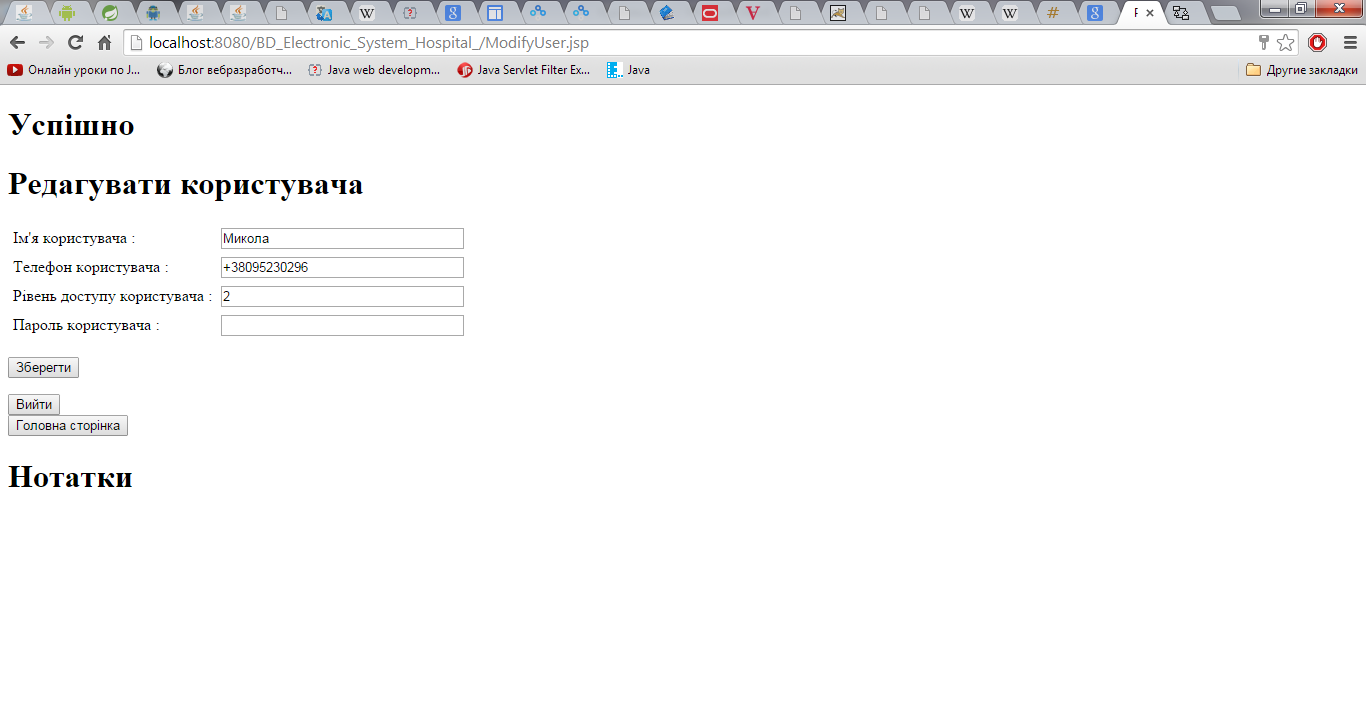


Рис. 4.10 Сторінка успішного внесення користувача

## Ілюстрація сценарію «Додати хворобу»

### Для внесення у базу даних нової хвороби адміністратору необхідно перейти на «Головну сторінку» Рис. 4.3.

### На «Головній сторінці» адміністратор повинен натиснути кнопку «Додати хворобу». Система пере направить адміністратора на сторінку «Додавання хвороби» Рис. 4.11.

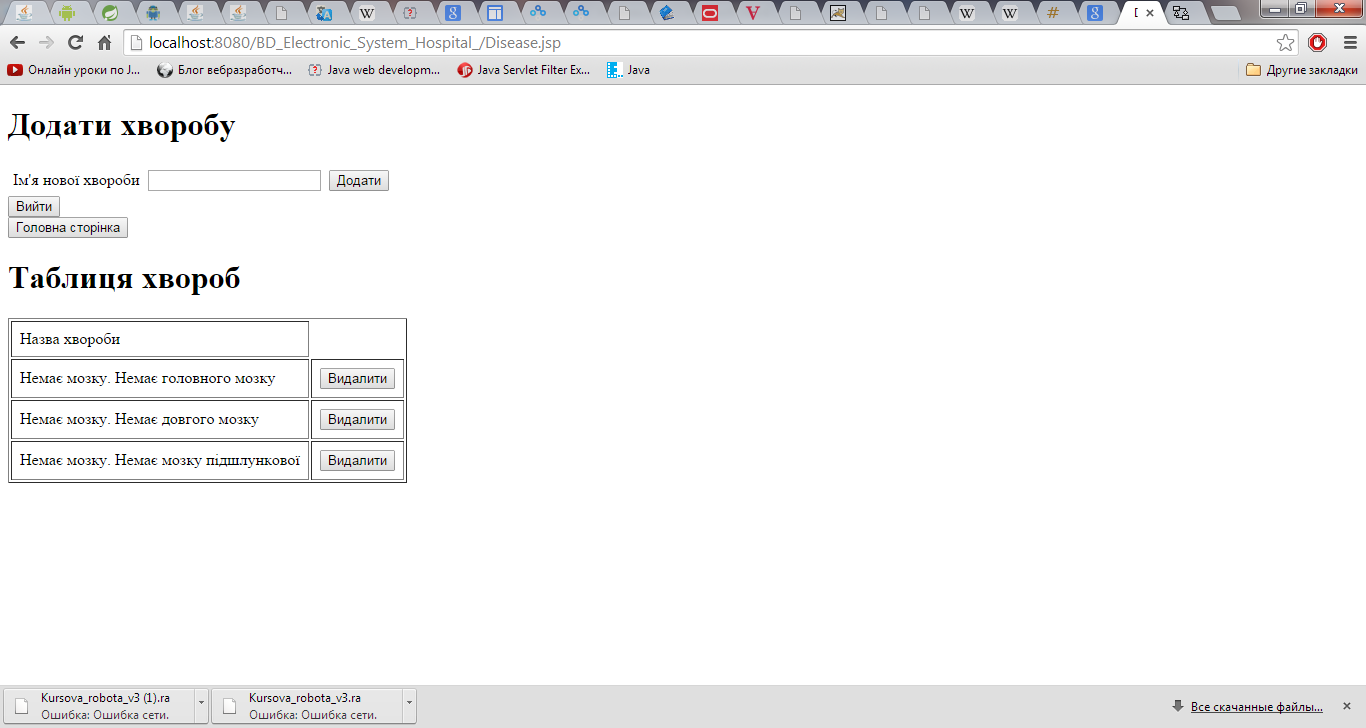


Рис. 4.11 Таблиця хвороб

### Адміністратору потрібно внести в поле назву хвороби і натиснути кнопку «Додати». У результаті успішного внесення даних система поверне адміністратору сторінку Рис. 4.12.

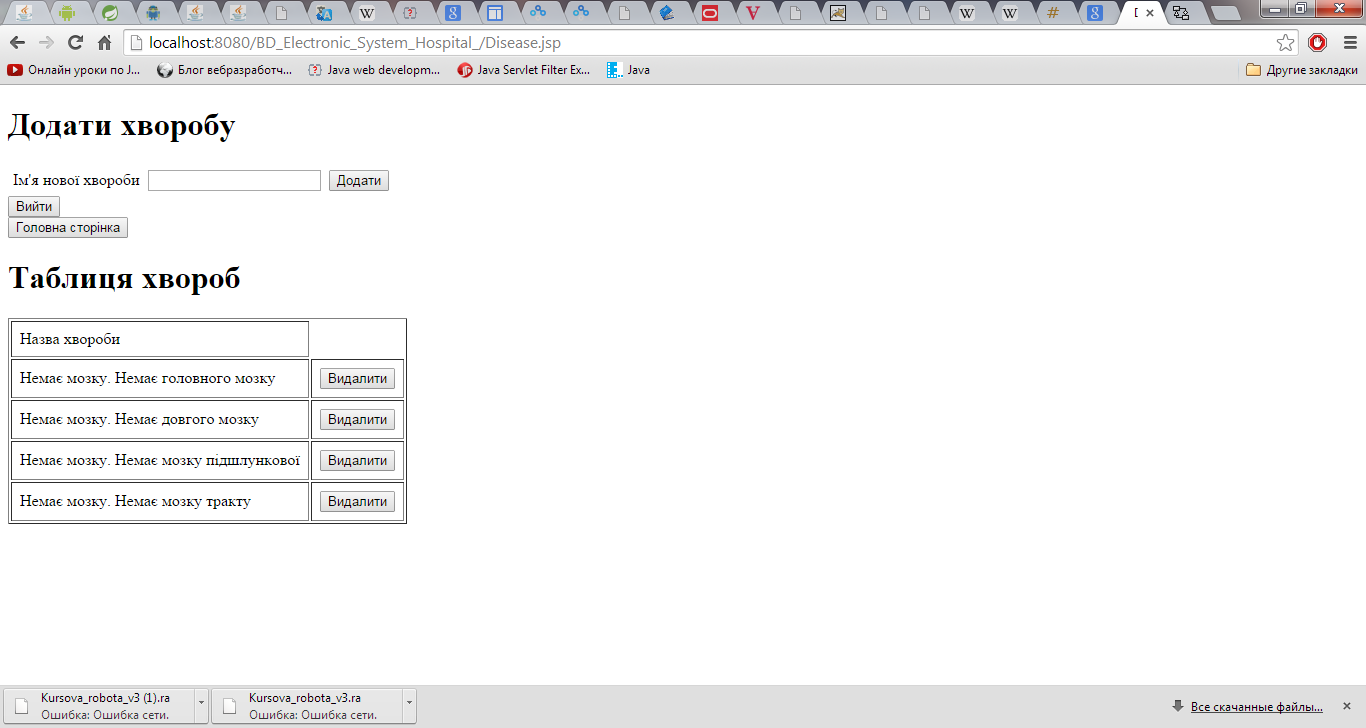


Рис. 4.12 Таблиця з доданою хворобою

## Ілюстрація сценарію «Видалення хвороби»

### Для видалення хвороби з бази адміністратору потрібно перейти на сторінку Рис.12 та на в проти видаляємо хвороби натиснути кнопку «Видалити». Система виконає операцію і перезавантажить сторінку дивися Рис. 4.13.

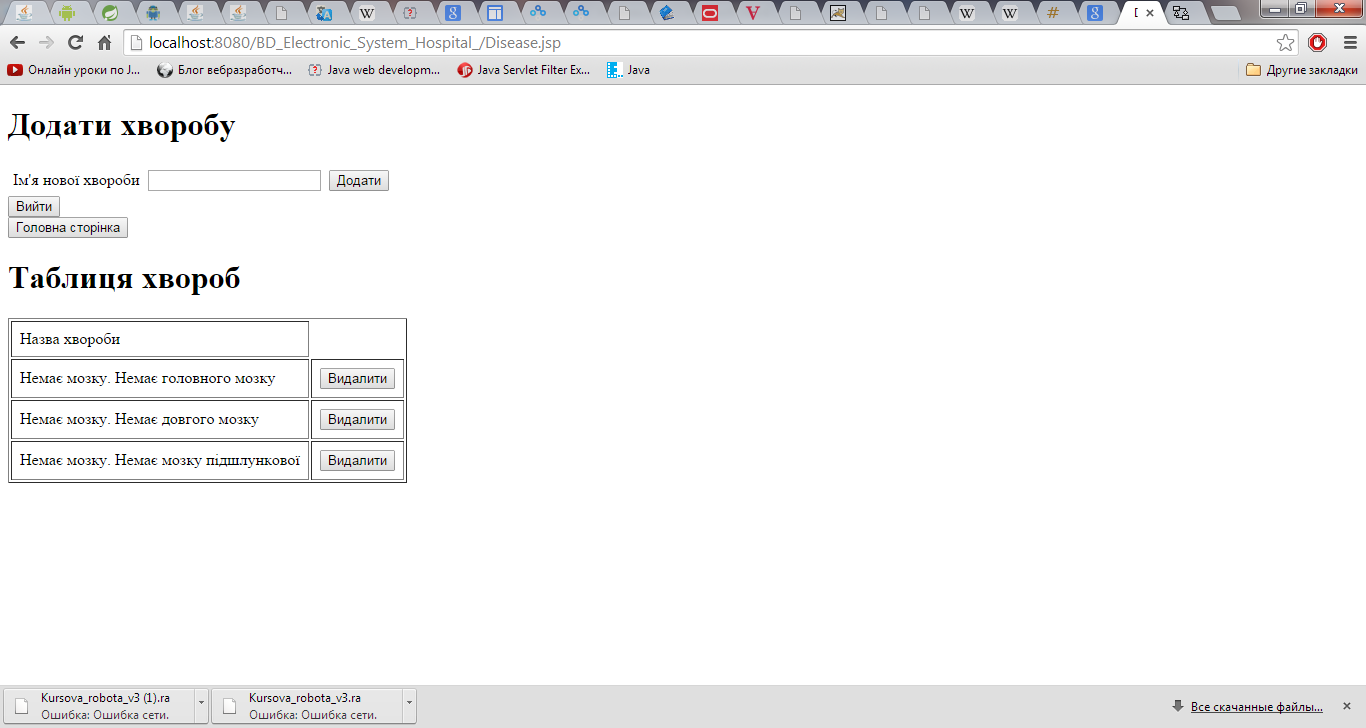


Рис. 4.13 Таблиця без видаленої хвороби

## Ілюстрація сценарію «Редагування власних даних»

### Для редагування власних даних адміністратор повинен перейти на сторінку Рис. 4.3 і натиснути кнопку «Налаштування аканту» після чого система відкриє сторінку Рис. 4.14.

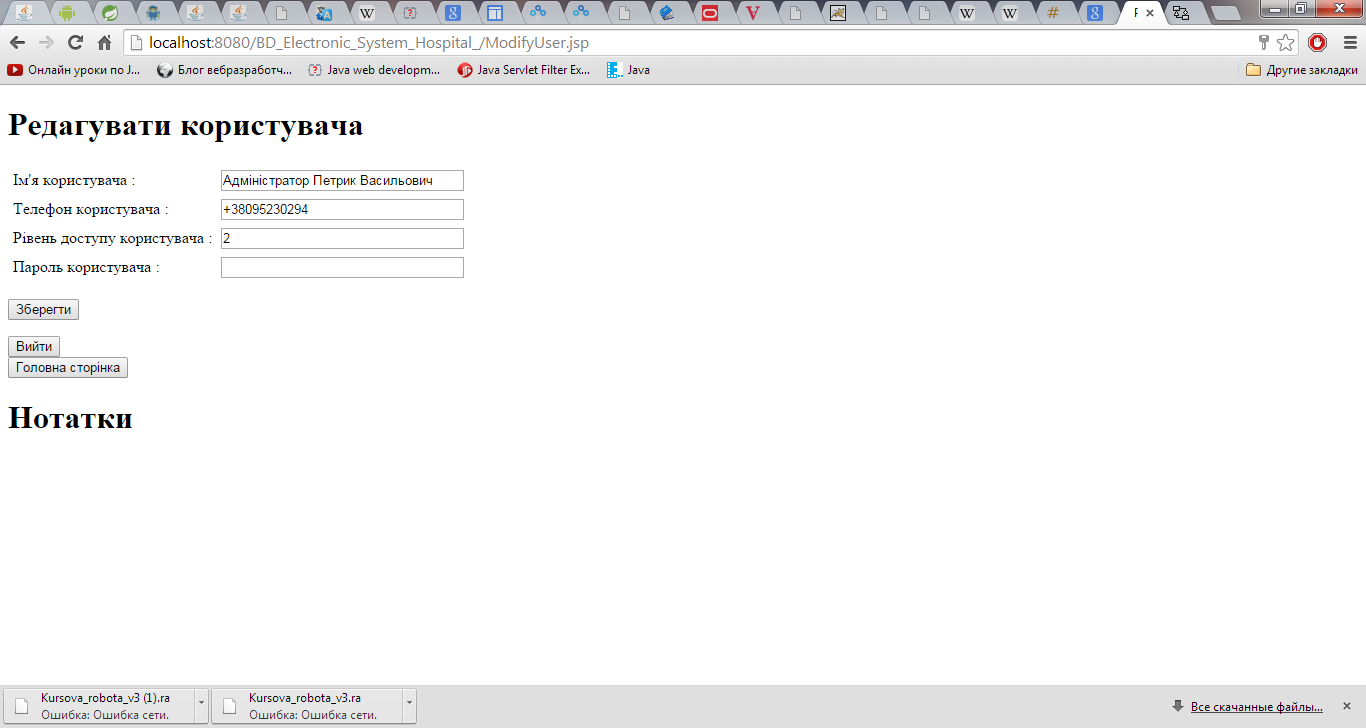


Рис. 4.14 Сторінка налаштувань

### Адміністратор повинен ввести редагує мі дані в поля та натиснути кнопку «Зберегти» після успішного виконання система пере направить на сторінку Рис. 4.15.

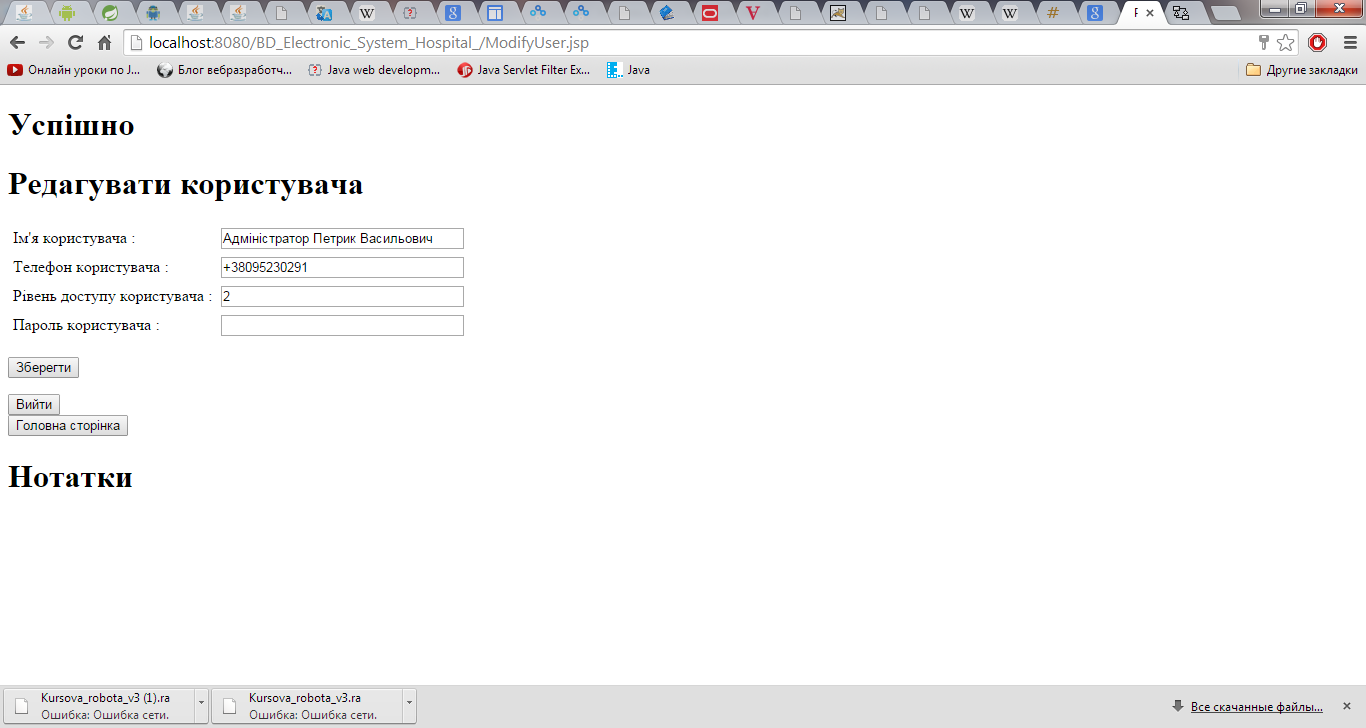


Рис. 4.15 Сторінка налаштувань після збереження даних

## Ілюстрація сценарію «Редагування користувача»

### Адміністратор повинен перейти на сторінку Рис. 4.5 і натиснути кнопку «Редагувати» після чого система відкриє сторінку редагування користувача (див. Рис. 4.16)

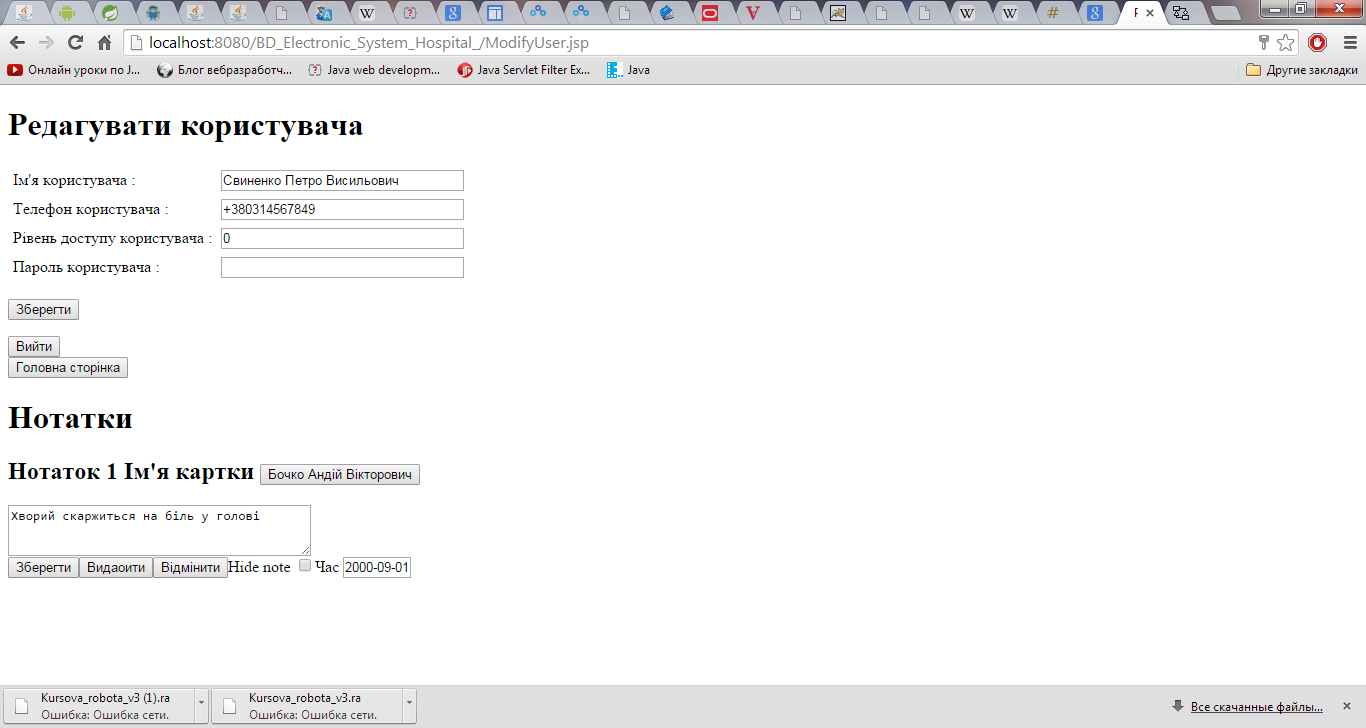


Рис. 4.16 Сторінка редагування користувача

### Адміністратор повинен внести зміни в поля і натиснути на кнопку «Зберегти» навпроти тієї форми, де було внесено зміни (див. Рис. 4.17)

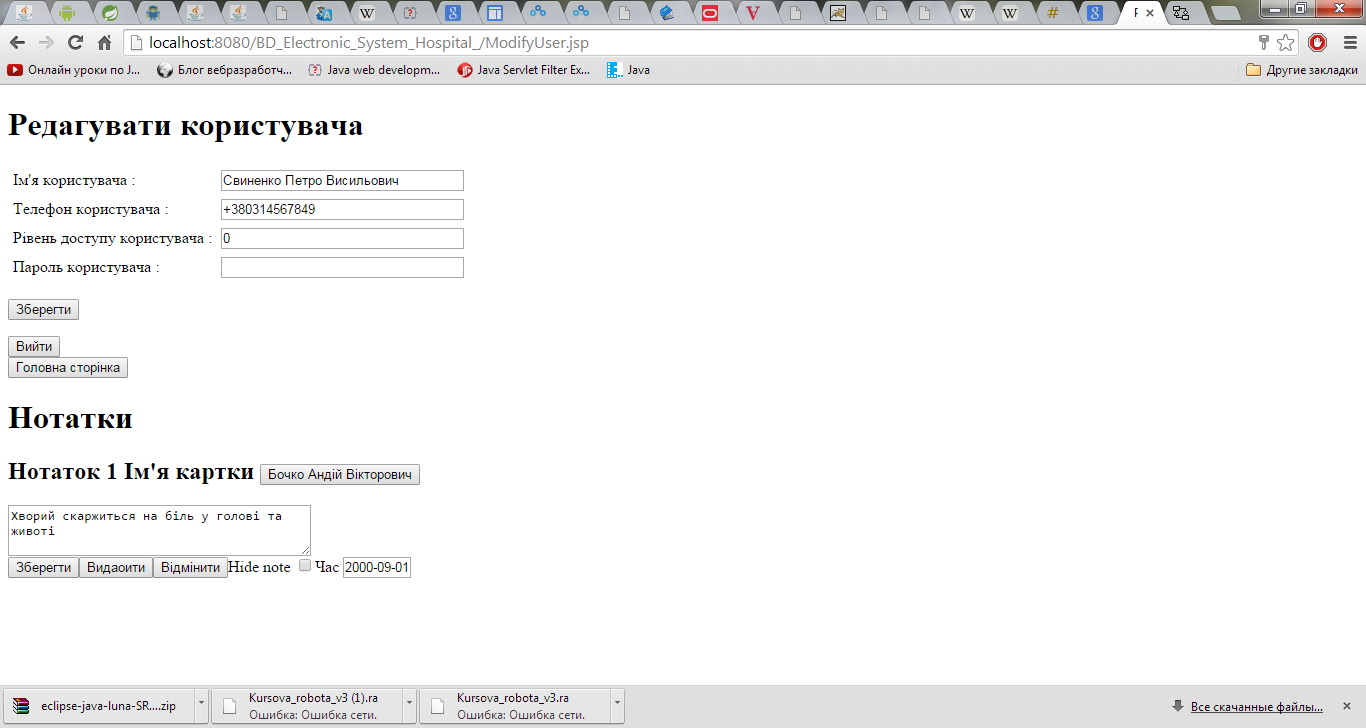


Рис. 4.17 Сторінка редагування користувача після збереження

## Ілюстрація сценарію «Редагування картки хворого»

### Адміністратор повинен перейти на сторінку Рис. 4.6 і натиснути кнопку «Редагувати» після чого система відкриє сторінку редагування користувача (див. Рис. 4.17)

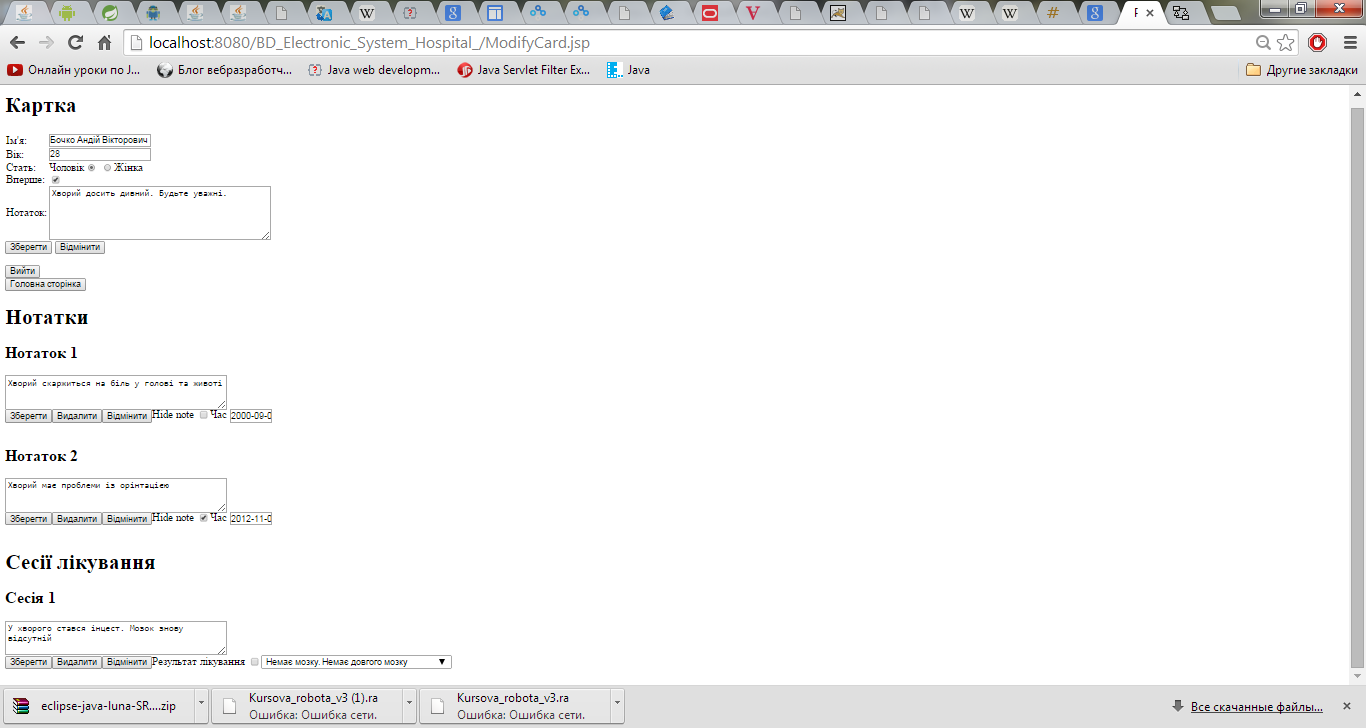


Рис. 4.17 Сторінка редагування картки

### Адміністратор повинен внести зміни в поля і натиснути на кнопку «Зберегти» навпроти тієї форми, де було внесено зміни (див. Рис. 4.18)

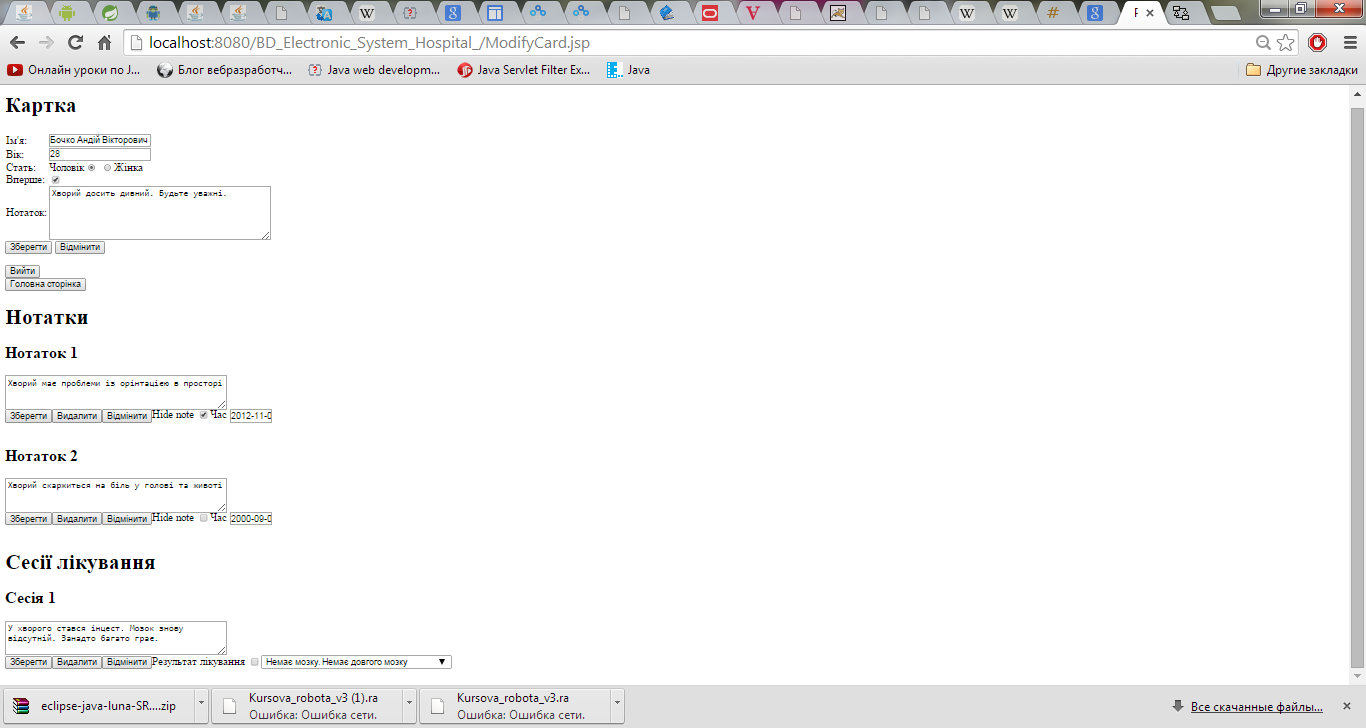


Рис. 4.18 Сторінка редагування картки після збереження

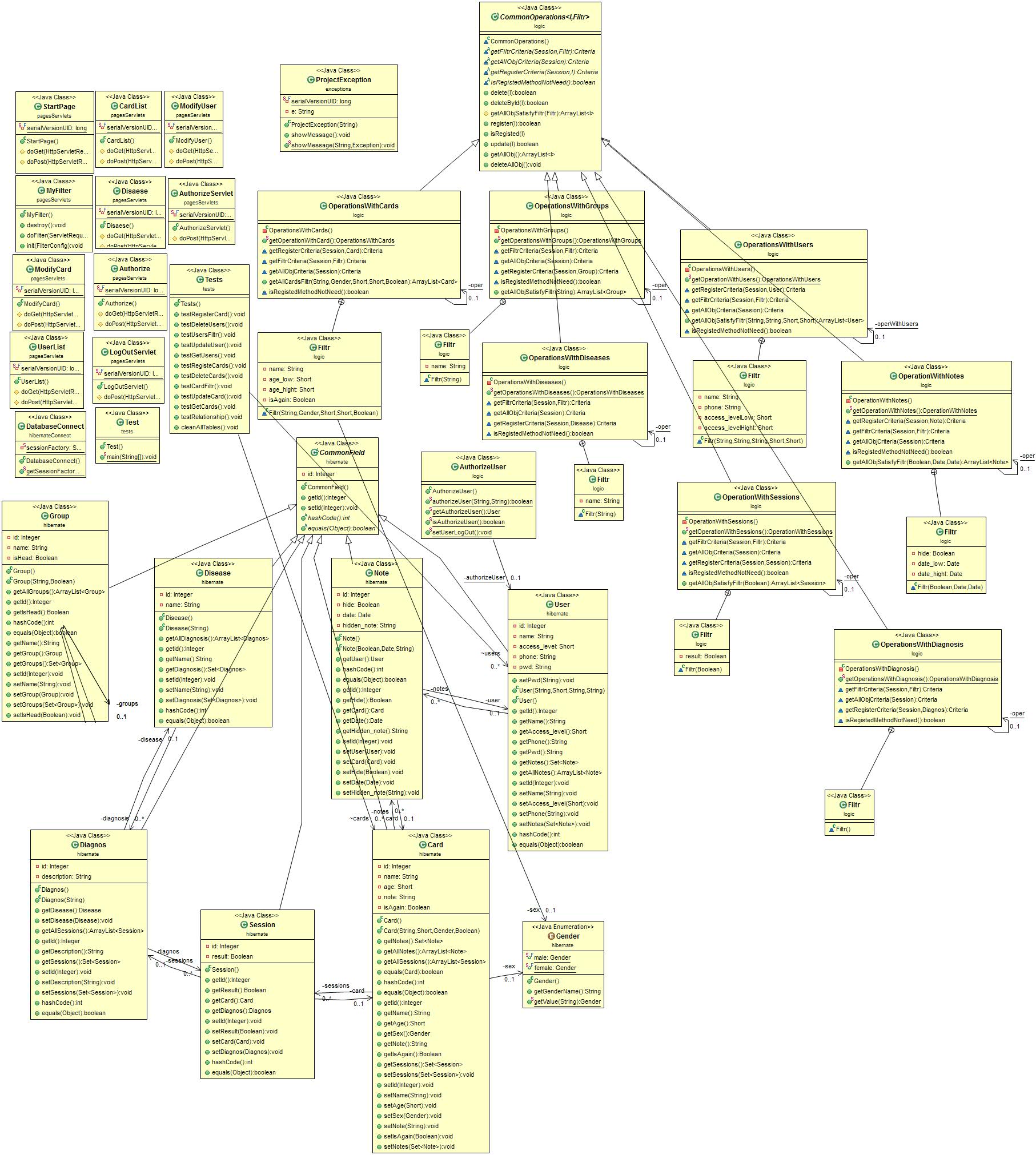
# СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Hibernate. Everything data. – Hibernate. Link: <http://hibernate.org/>
2. Apache Tomcat. Link: <http://tomcat.apache.org/>
3. JSP — Вікіпедія. Посилання: <https://uk.wikipedia.org/wiki/JSP/>
4. Сервлет — Вікіпедія. Посилання: <https://uk.wikipedia.org/wiki/Сервлет/>

# ДОДАТКИ

# ДОДАТОК А

Діаграма класів програми



# ДОДАТОК Б

SQL-код для створення таблиць бази даних

-- MySQL Administrator dump 1.4

--

-- ------------------------------------------------------

-- Server version 5.5.23

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!40101 SET NAMES utf8 \*/;

/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

--

-- Create schema hospital

--

**CREATE** DATABASE IF **NOT** **EXISTS** hospital;

USE hospital;

--

-- Definition of table `cards`

--

**DROP** **TABLE** IF **EXISTS** `cards`;

**CREATE** **TABLE** `cards` (

`id` **int**(11) **NOT** **NULL** AUTO\_INCREMENT,

`name` **varchar**(45) **NOT** **NULL**,

`age` **smallint**(5) unsigned **NOT** **NULL** **DEFAULT** '0',

`sex` enum('male','female') **NOT** **NULL**,

`note` text,

`isAgain` bit(1) **NOT** **NULL** **DEFAULT** b'0',

**PRIMARY** **KEY** (`id`)

) ENGINE=InnoDB **DEFAULT** CHARSET=utf8;

--

-- Dumping data for table `cards`

--

/\*!40000 ALTER TABLE `cards` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `cards` ENABLE KEYS \*/;

--

-- Definition of table `diagnosis`

--

**DROP** **TABLE** IF **EXISTS** `diagnosis`;

**CREATE** **TABLE** `diagnosis` (

`id` **int**(11) **NOT** **NULL** AUTO\_INCREMENT,

`description` **varchar**(100) **NOT** **NULL**,

`disease` **int**(11) **NOT** **NULL**,

**PRIMARY** **KEY** (`id`),

**KEY** `Diagnosis-Diseaser\_idx` (`disease`) **USING** BTREE,

**CONSTRAINT** `Diagnosis-GroupOwner` **FOREIGN** **KEY** (`disease`) **REFERENCES** `diagnosis` (`id`) **ON** **DELETE** **NO** **ACTION** **ON** **UPDATE** **NO** **ACTION**

) ENGINE=InnoDB **DEFAULT** CHARSET=utf8;

--

-- Dumping data for table `diagnosis`

--

/\*!40000 ALTER TABLE `diagnosis` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `diagnosis` ENABLE KEYS \*/;

--

-- Definition of table `diseases`

--

**DROP** **TABLE** IF **EXISTS** `diseases`;

**CREATE** **TABLE** `diseases` (

`id` **int**(10) unsigned **NOT** **NULL** AUTO\_INCREMENT,

`name` **varchar**(45) **NOT** **NULL**,

**PRIMARY** **KEY** (`id`)

) ENGINE=InnoDB **DEFAULT** CHARSET=utf8;

--

-- Dumping data for table `diseases`

--

/\*!40000 ALTER TABLE `diseases` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `diseases` ENABLE KEYS \*/;

--

-- Definition of table `groups`

--

**DROP** **TABLE** IF **EXISTS** `groups`;

**CREATE** **TABLE** `groups` (

`id` **int**(11) **NOT** **NULL** AUTO\_INCREMENT,

`name` **varchar**(45) **NOT** **NULL**,

`group\_id` **int**(11) **NOT** **NULL**,

`isHead` bit(1) **NOT** **NULL** **DEFAULT** b'1',

**PRIMARY** **KEY** (`id`),

**KEY** `GroupOwner\_idx` (`group\_id`),

**CONSTRAINT** `GroupOwner` **FOREIGN** **KEY** (`group\_id`) **REFERENCES** `groups` (`id`) **ON** **DELETE** **NO** **ACTION** **ON** **UPDATE** **NO** **ACTION**

) ENGINE=InnoDB **DEFAULT** CHARSET=utf8;

--

-- Dumping data for table `groups`

--

/\*!40000 ALTER TABLE `groups` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `groups` ENABLE KEYS \*/;

--

-- Definition of table `notes`

--

**DROP** **TABLE** IF **EXISTS** `notes`;

**CREATE** **TABLE** `notes` (

`id` **int**(10) **NOT** **NULL** AUTO\_INCREMENT,

`user\_id` **int**(10) **NOT** **NULL**,

`card\_id` **int**(10) **NOT** **NULL**,

`hide` bit(1) **NOT** **NULL** **DEFAULT** b'1',

`**date**` **date** **NOT** **NULL**,

`hidden\_note` text,

**PRIMARY** **KEY** (`id`),

**KEY** `Notes-Users\_idx` (`user\_id`),

**KEY** `Notes-Cars\_idx` (`card\_id`),

**CONSTRAINT** `Notes-Users` **FOREIGN** **KEY** (`user\_id`) **REFERENCES** `users` (`id`) **ON** **DELETE** **CASCADE** **ON** **UPDATE** **CASCADE**,

**CONSTRAINT** `Notes-Cards` **FOREIGN** **KEY** (`card\_id`) **REFERENCES** `cards` (`id`) **ON** **DELETE** **CASCADE** **ON** **UPDATE** **CASCADE**

) ENGINE=InnoDB **DEFAULT** CHARSET=utf8;

--

-- Dumping data for table `notes`

--

/\*!40000 ALTER TABLE `notes` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `notes` ENABLE KEYS \*/;

--

-- Definition of table `sessions`

--

**DROP** **TABLE** IF **EXISTS** `sessions`;

**CREATE** **TABLE** `sessions` (

`id` **int**(11) **NOT** **NULL** AUTO\_INCREMENT,

`card\_id` **int**(11) **NOT** **NULL**,

`result` bit(1) **NOT** **NULL** **DEFAULT** b'0',

`diagnosis\_id` **int**(11) **NOT** **NULL**,

**PRIMARY** **KEY** (`id`),

**KEY** `Sessions-Cards\_idx` (`card\_id`),

**KEY** `Sessions-Diagnosis\_idx` (`diagnosis\_id`),

**CONSTRAINT** `Sessions-Cards` **FOREIGN** **KEY** (`card\_id`) **REFERENCES** `cards` (`id`) **ON** **DELETE** **CASCADE** **ON** **UPDATE** **CASCADE**,

**CONSTRAINT** `Sessions-Diagnosis` **FOREIGN** **KEY** (`diagnosis\_id`) **REFERENCES** `diagnosis` (`id`) **ON** **DELETE** **CASCADE** **ON** **UPDATE** **CASCADE**

) ENGINE=InnoDB **DEFAULT** CHARSET=utf8;

--

-- Dumping data for table `sessions`

--

/\*!40000 ALTER TABLE `sessions` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `sessions` ENABLE KEYS \*/;

--

-- Definition of table `users`

--

**DROP** **TABLE** IF **EXISTS** `users`;

**CREATE** **TABLE** `users` (

`id` **int**(11) **NOT** **NULL** AUTO\_INCREMENT,

`name` **varchar**(45) **NOT** **NULL**,

`access\_level` **smallint**(6) **NOT** **NULL** **DEFAULT** '-1',

`phone` **varchar**(45) **DEFAULT** **NULL**,

`pwd` **varchar**(45) **NOT** **NULL**,

**PRIMARY** **KEY** (`id`)

) ENGINE=InnoDB **DEFAULT** CHARSET=utf8;

--

-- Dumping data for table `users`

--

/\*!40000 ALTER TABLE `users` DISABLE KEYS \*/;

/\*!40000 ALTER TABLE `users` ENABLE KEYS \*/;

/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;

/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

# ДОДАТОК В

Код програми

**package** exceptions;

/\*\*Клас для виведення повідомлень про виключення\*/

**public** **class** ProjectException **extends** Exception {

/\*\*Індефікатор класу виключень\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

/\*\*Супровідне повідомлення до виключення\*/

**private** String e;

/\*\*Створення об'єкту класу із вказанням супровідного повідомлення

\* **@param** e повідомлення про виключення\*/

**public** ProjectException(String e) {

**this**.e = e;

}

/\*\*Виведення повідомлення\*/

**public** **void** showMessage(){

System.*out*.print(e);

}

/\*\*Виведення повідомлення та логу виключення

\* **@param** message повідомлення про виключення

\* **@param** e лог виключення\*/

**public** **static** **void** showMessage(String message, Exception e) {

System.*out*.print(message + "Error log \n");

e.printStackTrace();

}

}

package hibernate;

import hibernateConnect.DatabaseConnect;

import java.util.ArrayList;

import java.util.Set;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.EnumType;

import javax.persistence.Enumerated;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.OneToMany;

import javax.persistence.Table;

import org.hibernate.Hibernate;

import org.hibernate.Session;

import org.hibernate.annotations.Type;

/\*\* Цей клас призначений для налаштування Hibernate для

\* коректної роботи з таблицею "cards" із бази даних "hospital" \*/

@Entity

@Table(name="cards")

public class Card extends CommonField {

/\*\*Опис поля "id" таблиці "cards"\*/

@Id

@GeneratedValue

@Column(name = "id")

private Integer id;

/\*\*Опис поля "name" таблиці "cards"\*/

@Column(name = "name")

private String name;

/\*\*Опис поля "age" таблиці "cards"\*/

@Column(name = "age")

private Short age;

/\*\*Опис поля "sex" таблиці "cards"\*/

@Column(name = "sex")

@Enumerated(EnumType.STRING)

private Gender sex;

/\*\*Опис поля "note" таблиці "cards"\*/

@Column(name = "note")

@Type(type = "text")

private String note;

/\*\*Опис поля "isAgain" таблиці "cards"\*/

@Column(name = "isAgain")

@Type(type = "boolean")

private Boolean isAgain;

/\*\*Опис зміної масиву даних для витягування всіх сесій, які стосуються даної картки,

\* із зв'язаної таблиці "sessions" по індефікатору "card\_id", який буде співпадати

\* з індефікатором "id" поля з таблиці "cards". Відношення один до багатьох

\* \*/

@OneToMany(fetch = FetchType.LAZY, mappedBy = "card", cascade = CascadeType.ALL)

private Set<hibernate.Session> sessions;

/\*\*Опис зміної масиву даних для витягування всіх нотатків, які стосуються даної картки,

\* із зв'язаної таблиці "sessions" по індефікатору "card\_id", який буде співпадати з

\* індефікатором "id" поля з таблиці "cards". Відношення один до багатьох

\* \*/

@OneToMany(fetch = FetchType.LAZY, mappedBy="card")

private Set<Note> notes;

/\*\*Створення об'єкта без налаштувань його полів \*/

public Card() {

}

/\*\*Створення об'єкта класа із заданням полів

\* @param name ім'я пацієнта цієї картки

\* @param age вік пацієнта цієї картки

\* @param sex стать пацієнта цієї картки

\* @param isAgain чи вперше пацієнт

\* \*/

public Card(String name, Short age, Gender sex, Boolean isAgain) {

this.age = age;

this.name = name;

this.sex = sex;

this.isAgain = isAgain;

}

/\*\*Метод повертає структуру, що містить масив нотатків, які стосуються цієї

\* картки. При умові, що ці дані вже завантажені з бази даних в програму,

\* інакше отримаєте помилку

\* @return колекцію сесій\*/

public Set<Note> getNotes() {

return notes;

}

/\*\*

\* Цей проект використовує "Lazy initialization" - це означає, що дані із зв'язаних

\* таблиць(нотатки) не завантажуються автоматично, тому в цьому класі два метода для

\* отримання даних. Перший метод "getNotes" повертає структуру даних, якщо ці дані

\* вже завантажені, другий метод "getAllNotes" створює запит до бази даних і

\* завантажує з неї необхідні дані і повертає структуру з цими даними, що

\* стосуються поточної картки.

\* @return колекцію нотатків

\*/

public ArrayList<Note> getAllNotes() {

ArrayList<Note> listNotes;

Session session = DatabaseConnect.getSessionFactory().openSession();

Card card = (Card) session.load(Card.class, this.id);

Hibernate.initialize(card.notes);

listNotes = new ArrayList<>(card.getNotes());

session.close();

return listNotes;

}

/\*\*

\* Цей проект використовує "Lazy initialization" - це означає, що дані із зв'язаних

\* таблиць(сесії) не завантажуються автоматично, тому в цьому класі два метода для

\* отримання даних. Перший метод "getSessions" повертає структуру даних, якщо ці

\* дані вже завантажені, другий метод "getAllSessions" створює запит до бази даних

\* і завантажує з неї необхідні дані і повертає структуру з цими даними, що

\* стосуються поточної картки.

\* @return колекцію сесій

\*/

public ArrayList<hibernate.Session> getAllSessions() {

ArrayList<hibernate.Session> listSessions;

Session session = DatabaseConnect.getSessionFactory().openSession();

Card card = (Card) session.load(Card.class, this.id);

Hibernate.initialize(card.sessions);

listSessions = new ArrayList<>(card.getSessions());

session.close();

return listSessions;

}

/\*\*Порівнює дві картки

\* @param card картка з якою треба порівняти

\* @return true якщо рівні або false якщо ні\*/

public boolean equals(Card card) {

if ( card.age.equals(age) && card.name.equals(name) && card.sex == sex

&& card.isAgain.equals(isAgain) && ((note == null && note == card.note)

|| card.note.equals(note)))

return true;

else return false;

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + ((age == null) ? 0 : age.hashCode());

result = prime \* result

+ ((sessions == null) ? 0 : sessions.hashCode());

result = prime \* result + ((id == null) ? 0 : id.hashCode());

result = prime \* result + ((isAgain == null) ? 0 : isAgain.hashCode());

result = prime \* result + ((name == null) ? 0 : name.hashCode());

result = prime \* result + ((note == null) ? 0 : note.hashCode());

result = prime \* result + ((notes == null) ? 0 : notes.hashCode());

result = prime \* result + ((sex == null) ? 0 : sex.hashCode());

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Card other = (Card) obj;

if (age == null) {

if (other.age != null)

return false;

} else if (!age.equals(other.age))

return false;

if (id == null) {

if (other.id != null)

return false;

} else if (!id.equals(other.id))

return false;

if (isAgain == null) {

if (other.isAgain != null)

return false;

} else if (!isAgain.equals(other.isAgain))

return false;

if (name == null) {

if (other.name != null)

return false;

} else if (!name.equals(other.name))

return false;

if (note == null) {

if (other.note != null)

return false;

} else if (!note.equals(other.note))

return false;

if (notes == null) {

if (other.notes != null)

return false;

} else if (!notes.equals(other.notes))

return false;

if (sessions == null) {

if (other.sessions != null)

return false;

} else if (!sessions.equals(other.sessions))

return false;

if (sex != other.sex)

return false;

return true;

}

/\*\*@return повертає ціле число, яке є індефікатором цього об'єкта

\* в базі даних\*/

public Integer getId() {

return id;

}

/\*\*@return повертає ім'я пацієнта картки\*/

public String getName() {

return name;

}

/\*\*@return повертає вік пацієнта картки\*/

public Short getAge() {

return age;

}

/\*\*@return повертає стать пацієнта цієї картки\*/

public Gender getSex() {

return sex;

}

/\*\*@return повертає припис лікаря до цієї картки\*/

public String getNote() {

return note;

}

/\*\*@return повертає true, якщо пацієнт не вперше і

\* відповідно false, якщо вперше \*/

public Boolean getIsAgain() {

return isAgain;

}

/\*\*Метод повертає структуру, що містить масив сесій, які стосуються цієї картки.

\* При умові, що ці дані вже завантажені з бази даних в програму,

\* інакше отримаєте помилку

\* @return повертає масив сесій, які стосуються цієї картки

\* \*/

public Set<hibernate.Session> getSessions() {

return sessions;

}

/\*\*Встановлює для картки параметр

\* @param sessions колекція сесій\*/

public void setSessions(Set<hibernate.Session> sessions) {

this.sessions = sessions;

}

/\*\*Встановлює для картки параметр

\* @param id індетифікатор для картки в базі даних\*/

public void setId(Integer id) {

this.id = id;

}

/\*\*Встановлює для картки параметр

\* @param name ім'я пацієнта цієї картки\*/

public void setName(String name) {

this.name = name;

}

/\*\*Встановлює для картки параметр

\* @param age вік пацієнта цієї картки\*/

public void setAge(Short age) {

this.age = age;

}

/\*\*Встановлює для картки параметр

\* @param sex встановлює стать пацієнта цієї картки\*/

public void setSex(Gender sex) {

this.sex = sex;

}

/\*\*Встановлює для картки параметр

\* @param note встановлює записЫ до картки\*/

public void setNote(String note) {

this.note = note;

}

/\*\*Встановлює для картки параметр

\* @param isAgain встановлює вперше чи ні\*/

public void setIsAgain(Boolean isAgain) {

this.isAgain = isAgain;

}

/\*\*Встановлює для картки параметр

\* @param notes встановлює масив нотатків до картки\*/

public void setNotes(Set<Note> notes) {

this.notes = notes;

}

}

**package** hibernate;

/\*\*Клас, що містить спільне поле для всіх таблиць\*/

**public** **abstract** **class** CommonField {

/\*\*Індефікатор в таблиці\*/

**private** Integer id;

/\*\***@return** індефікатор даних в таблиці\*/

**public** Integer getId() {

**return** id;

}

/\*\***@param** id встановити індефікатор для даних в таблиці\*/

**public** **void** setId(Integer id) {

**this**.id = id;

}

/\*\*Отримання хеш-коду даного об'єкта

\* **@return** повертає ціле число, яке є хеш-кодом\*/

**abstract** **public** **int** hashCode();

/\*\*Порівнює два діагнози

\* **@param** obj діагноз з яким треба порівняти

\* **@return** true якщо рівні або false якщо ні\*/

**abstract** **public** **boolean** equals(Object obj);

}

**package** hibernate;

**import** hibernateConnect.DatabaseConnect;

**import** java.util.ArrayList;

**import** java.util.Set;

**import** javax.persistence.CascadeType;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.FetchType;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.Id;

**import** javax.persistence.JoinColumn;

**import** javax.persistence.ManyToOne;

**import** javax.persistence.OneToMany;

**import** javax.persistence.Table;

**import** org.hibernate.Hibernate;

**import** org.hibernate.Session;

**import** org.hibernate.annotations.GenericGenerator;

/\*\* Цей клас призначений для налаштування Hibernate для

\* коректної роботи з таблицею "diagnosis" із бази даних "hospital" \*/

@Entity

@Table(name="diagnosis")

**public** **class** Diagnos **extends** CommonField {

/\*\*Індефікатор даних в таблиці "diagnosis"\*/

@Id

@GeneratedValue(generator="increment")

@GenericGenerator(name="increment", strategy = "increment")

@Column(name="id")

**private** Integer id;

/\*\*Приспис до діагнозу в таблиці "diagnosis"\*/

@Column(name = "description")

**private** String description;

/\*\*Опис зміної масиву даних для витягування всіх сесій, які стосуються даного діагнозу,

\* із зв'язаної таблиці "sessions" по індефікатору "diagnosis\_id", який буде співпадати

\* з індефікатором "id" поля з таблиці "diagnos". Відношення один до багатьох

\* \*/

@OneToMany(fetch = FetchType.LAZY, mappedBy="diagnos", cascade = CascadeType.ALL)

**private** Set<hibernate.Session> sessions;

/\*\*Зв'язний параметр з таблицею "diseases". Відношення багато до одного\*/

@ManyToOne

@JoinColumn(name="disease")

**private** Disease disease;

/\*\*Створення об'єкта класа\*/

**public** Diagnos() {

}

/\*\*Створення об'єкта класа із параметрами

\* **@param** description припис до діагнозу\*/

**public** Diagnos(String description) {

**this**.description = description;

}

/\*\***@return** повертає об'єкт хвороби\*/

**public** Disease getDisease() {

**return** disease;

}

/\*\***@param** disease встановлення хвороби для діагнозу\*/

**public** **void** setDisease(Disease disease) {

**this**.disease = disease;

}

/\*\*

\* Цей проект використовує "Lazy initialization" - це означає, що дані із зв'язаних

\* таблиць(сесії) не завантажуються автоматично, тому в цьому класі два метода для

\* отримання даних. Перший метод "getSessions" повертає структуру даних, якщо ці дані

\* вже завантажені, другий метод "getAllSessions" створює запит до бази даних і

\* завантажує з неї необхідні дані і повертає структуру з цими даними, що

\* стосуються поточного діагнозу.

\* **@return** колекцію сесій

\*/

**public** ArrayList<hibernate.Session> getAllSessions() {

ArrayList<hibernate.Session> listSessions;

Session session = DatabaseConnect.getSessionFactory().openSession();

Diagnos diagnos = (Diagnos) session.load(Diagnos.**class**, **this**.id);

Hibernate.initialize(diagnos.sessions);

listSessions = **new** ArrayList<>(diagnos.getSessions());

session.close();

**return** listSessions;

}

/\*\***@return** повертає індефікатор в базі даних\*/

**public** Integer getId() {

**return** id;

}

/\*\***@return** повертає припис до діагнозу\*/

**public** String getDescription() {

**return** description;

}

/\*\***@return** повертає колекцію сесій\*/

**public** Set<hibernate.Session> getSessions() {

**return** sessions;

}

/\*\***@param** id встановлює індефікатор в базі даних\*/

**public** **void** setId(Integer id) {

**this**.id = id;

}

/\*\***@param** description встановлює припис до діагнозу\*/

**public** **void** setDescription(String description) {

**this**.description = description;

}

/\*\***@param** sessions встановлює колекцію сесій для діагнозу\*/

**public** **void** setSessions(Set<hibernate.Session> sessions) {

**this**.sessions = sessions;

}

/\*\*Отримання хеш-коду даного об'єкта

\* **@return** повертає ціле число, яке є хеш-кодом\*/

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result

+ ((description == **null**) ? 0 : description.hashCode());

result = prime \* result + ((id == **null**) ? 0 : id.hashCode());

**return** result;

}

/\*\*Порівнює два об'єкта

\* **@param** obj об'єкт з яким треба порівняти

\* **@return** true якщо рівні або false якщо ні\*/

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Diagnos other = (Diagnos) obj;

**if** (description == **null**) {

**if** (other.description != **null**)

**return** **false**;

} **else** **if** (!description.equals(other.description))

**return** **false**;

**if** (id == **null**) {

**if** (other.id != **null**)

**return** **false**;

} **else** **if** (!id.equals(other.id))

**return** **false**;

**return** **true**;

}

}

package hibernate;

import hibernateConnect.DatabaseConnect;

import java.util.ArrayList;

import java.util.Set;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.OneToMany;

import javax.persistence.Table;

import org.hibernate.Hibernate;

import org.hibernate.Session;

import org.hibernate.annotations.GenericGenerator;

/\*\* Цей клас призначений для налаштування Hibernate для

\* коректної роботи з таблицею "diseases" із бази даних "hospital" \*/

@Entity

@Table(name="diseases")

public class Disease extends CommonField {

/\*\*Індефікатор даних в таблиці "diseases"\*/

@Id

@GeneratedValue(generator="increment")

@GenericGenerator(name="increment", strategy = "increment")

@Column(name = "id")

private Integer id;

/\*\*Назва звороби в таблиці "diseases"\*/

@Column(name = "name")

private String name;

/\*\*Опис зміної масиву даних для витягування всіх діагнозів, які стосуються даної хвороби,

\* із зв'язаної таблиці "diagnosis" по індефікатору "disease", який буде співпадати

\* з індефікатором "id" поля з таблиці "disease". Відношення один до багатьох

\* \*/

@OneToMany(fetch = FetchType.LAZY, mappedBy="disease")

private Set<Diagnos> diagnosis;

/\*\*Створення об'єкта класа\*/

public Disease() {

}

/\*\*Створення об'єкта класа із параметрами

\* @param name ім'я хвороби\*/

public Disease(String name) {

this.name = name;

}

/\*\*

\* Цей проект використовує "Lazy initialization" - це означає, що дані із зв'язаних

\* таблиць(діагноз) не завантажуються автоматично, тому в цьому класі два метода для

\* отримання даних. Перший метод "getDiagnosis" повертає структуру даних, якщо ці дані

\* вже завантажені, другий метод "getAllDiagnosis" створює запит до бази даних і

\* завантажує з неї необхідні дані і повертає структуру з цими даними, що

\* стосуються поточної хвороби.

\* @return колекцію діагнозів

\*/

public ArrayList<Diagnos> getAllDiagnosis() {

ArrayList<Diagnos> listDiagnosis;

Session session = DatabaseConnect.getSessionFactory().openSession();

Disease disease = (Disease) session.load(Disease.class, this.id);

Hibernate.initialize(disease.diagnosis);

listDiagnosis = new ArrayList<>(disease.getDiagnosis());

session.close();

return listDiagnosis;

}

public Integer getId() {

return id;

}

/\*\*@return повертає назву хвороби\*/

public String getName() {

return name;

}

/\*\*@return колекцію діагнозів\*/

public Set<Diagnos> getDiagnosis() {

return diagnosis;

}

public void setId(Integer id) {

this.id = id;

}

/\*\*@param name встановити ім'я хвороби\*/

public void setName(String name) {

this.name = name;

}

/\*\*@param diagnosis встановити колекцію діагнозів для хвороби\*/

public void setDiagnosis(Set<Diagnos> diagnosis) {

this.diagnosis = diagnosis;

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + ((id == null) ? 0 : id.hashCode());

result = prime \* result + ((name == null) ? 0 : name.hashCode());

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Disease other = (Disease) obj;

if (id == null) {

if (other.id != null)

return false;

} else if (!id.equals(other.id))

return false;

if (name == null) {

if (other.name != null)

return false;

} else if (!name.equals(other.name))

return false;

return true;

}

}

**package** hibernate;

/\*\*Допоміжний клас-перечислення для встановлення статі\*/

**public** **enum** Gender {

*male*,

*female*;

/\*\***@return** строку, яка є еквівалентом значення поточного об'єкту\*/

**public** String getGenderName() {

**switch** (**this**) {

**case** *male* : **return** "Man";

**case** *female* : **return** "Woman";

**default** : **return** **null**;

}

}

/\*\***@param** p переданий об'єкт

\* **@return** строку, яка є еквівалентом значенню переданого об'єкту\*/

**public** **static** Gender getValue(String p) {

**switch** (p) {

**case** "Man" : **return** Gender.*male*;

**case** "Woman" : **return** Gender.*female*;

**default** : **return** **null**;

}

}

};

package hibernate;

import hibernateConnect.DatabaseConnect;

import java.util.ArrayList;

import java.util.Set;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import javax.persistence.OneToMany;

import javax.persistence.Table;

import org.hibernate.Hibernate;

import org.hibernate.Session;

import org.hibernate.annotations.GenericGenerator;

import org.hibernate.annotations.Type;

/\*\* Цей клас призначений для налаштування Hibernate для

\* коректної роботи з таблицею "groups" із бази даних "hospital" \*/

@Entity

@Table(name="groups")

public class Group extends CommonField {

/\*\*Індефікатор даних в таблиці "diagnosis"\*/

@Id

@GeneratedValue(generator="increment")

@GenericGenerator(name="increment", strategy = "increment")

@Column(name = "id")

private Integer id;

/\*\*Поле ім'я групи в таблиці "groups"\*/

@Column(name = "name")

private String name;

/\*\*Поле статусу групи в таблиці "groups"\*/

@Column(name = "isHead")

@Type(type = "boolean")

private Boolean isHead;

/\*\*Поле групи в таблиці "groups"\*/

@ManyToOne

@JoinColumn(name="group\_id")

private Group group;

/\*\*Опис зміної масиву даних для витягування всіх груп, які стосуються даної групи,

\* із зв'язаної таблиці "groups" по індефікатору "group\_id", який буде співпадати

\* з індефікатором "id" поля з таблиці "groups". Відношення один до багатьох

\* \*/

@OneToMany(fetch = FetchType.LAZY, mappedBy="group")

private Set<Group> groups;

/\*\*Створення об'єкта класа\*/

public Group() {

}

/\*\*Створення об'єкта класа із параметрами

\* @param name назва групи

\* @param isHead при встановлені в true ця група важається головою,

\* при false не головою групи\*/

public Group(String name, Boolean isHead) {

this.name = name;

this.isHead = isHead;

}

/\*\*

\* Цей проект використовує "Lazy initialization" - це означає, що дані із зв'язаних

\* таблиць(група) не завантажуються автоматично, тому в цьому класі два метода для

\* отримання даних. Перший метод "getGroups" повертає структуру даних, якщо ці дані

\* вже завантажені, другий метод "getAllGroups" створює запит до бази даних і

\* завантажує з неї необхідні дані і повертає структуру з цими даними, що

\* стосуються поточної групи.

\* @return колекцію груп

\*/

public ArrayList<Group> getAllGroups() {

ArrayList<Group> listGroups;

Session session = DatabaseConnect.getSessionFactory().openSession();

Group group = (Group) session.load(Group.class, this.id);

Hibernate.initialize(group.groups);

listGroups = new ArrayList<>(group.getGroups());

session.close();

if ( listGroups.isEmpty() || listGroups.get(0).id.equals(this.id))

return null;

else return listGroups;

}

public Integer getId() {

return id;

}

/\*\*@return повертає true якщо група є початком, false якщо ні\*/

public Boolean getIsHead() {

return isHead;

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + ((id == null) ? 0 : id.hashCode());

result = prime \* result + ((name == null) ? 0 : name.hashCode());

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Group other = (Group) obj;

if (id == null) {

if (other.id != null)

return false;

} else if (!id.equals(other.id))

return false;

if (name == null) {

if (other.name != null)

return false;

} else if (!name.equals(other.name))

return false;

return true;

}

/\*\*@return повертає назву групи\*/

public String getName() {

return name;

}

/\*\*@return повертає об'єкт групи\*/

public Group getGroup() {

return group;

}

/\*\*@return повертає колекцію груп\*/

public Set<Group> getGroups() {

return groups;

}

public void setId(Integer id) {

this.id = id;

}

/\*\*@param name встановлення назву для групи\*/

public void setName(String name) {

this.name = name;

}

/\*\*@param group встановлення групу для поточної групи\*/

public void setGroup(Group group) {

this.group = group;

}

/\*\*@param groups встановлення колекцію груп\*/

public void setGroups(Set<Group> groups) {

this.groups = groups;

}

/\*\*@param isHead встановлення голову групи\*/

public void setIsHead(Boolean isHead) {

this.isHead = isHead;

}

}

package hibernate;

import java.sql.Date;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import javax.persistence.Table;

import org.hibernate.annotations.Type;

/\*\* Цей клас призначений для налаштування Hibernate для

\* коректної роботи з таблицею "notes" із бази даних "hospital" \*/

@Entity

@Table(name="notes")

public class Note extends CommonField {

/\*\*Індефікатор даних в таблиці "notes"\*/

@Id

@GeneratedValue

@Column(name = "id")

private Integer id;

/\*\*Зв'язний параметр з таблицею "cards". Відношення багато до одного\*/

@ManyToOne

@JoinColumn(name = "card\_id")

private Card card;

/\*\*Зв'язний параметр з таблицею "user". Відношення багато до одного\*/

@ManyToOne

@JoinColumn(name="user\_id")

private User user;

/\*\*Поле для встановлення режиму видемості нотатку\*/

@Column(name = "hide")

private Boolean hide;

/\*\*Поле часу створення нотатку\*/

@Column(name = "date")

@Type(type = "date")

private Date date;

/\*\*Поле текстового повідомлення нотатку\*/

@Column(name = "hidden\_note")

@Type(type = "text")

private String hidden\_note;

/\*\*Створення об'єкта класа\*/

public Note() {

}

/\*\*Створення об'єкта класа із параметрами

\* @param hide режим видимості

\* @param date час створення нотатку

\* @param hidden\_note текст нотатку\*/

public Note(Boolean hide, Date date, String hidden\_note) {

this.hide = hide;

this.date = date;

this.hidden\_note = hidden\_note;

}

/\*\*@return повертає об'єкт користувача\*/

public User getUser() {

return user;

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + ((date == null) ? 0 : date.hashCode());

result = prime \* result

+ ((hidden\_note == null) ? 0 : hidden\_note.hashCode());

result = prime \* result + ((hide == null) ? 0 : hide.hashCode());

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Note other = (Note) obj;

if (date == null) {

if (other.date != null)

return false;

} else if (!date.equals(other.date))

return false;

if (hidden\_note == null) {

if (other.hidden\_note != null)

return false;

} else if (!hidden\_note.equals(other.hidden\_note))

return false;

if (hide == null) {

if (other.hide != null)

return false;

} else if (!hide.equals(other.hide))

return false;

return true;

}

public Integer getId() {

return id;

}

/\*\*@return повертає режим видемості нотатку\*/

public Boolean getHide() {

return hide;

}

/\*\*@return повертає об'єкт карти\*/

public Card getCard() {

return card;

}

/\*\*@return повертає об'єкт часу\*/

public Date getDate() {

return date;

}

/\*\*@return повертає текст нотатку\*/

public String getHidden\_note() {

return hidden\_note;

}

public void setId(Integer id) {

this.id = id;

}

/\*\*@param user встановлення об'єкта користувача\*/

public void setUser(User user) {

this.user = user;

}

/\*\*@param card встановлення об'єкта карти\*/

public void setCard(Card card) {

this.card = card;

}

/\*\*@param hide встановлення видимості нотатку, якщо true то нотаток

\* бачить лікарь, який його створив та адміністратори\*/

public void setHide(Boolean hide) {

this.hide = hide;

}

/\*\*@param date встановлення об'єкт часу\*/

public void setDate(Date date) {

this.date = date;

}

/\*\*@param hidden\_note встановлення текст нотатку\*/

public void setHidden\_note(String hidden\_note) {

this.hidden\_note = hidden\_note;

}

}

package hibernate;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import javax.persistence.Table;

import org.hibernate.annotations.Type;

/\*\* Цей клас призначений для налаштування Hibernate для

\* коректної роботи з таблицею "sessions" із бази даних "hospital" \*/

@Entity

@Table(name="sessions")

public class Session extends CommonField {

/\*\*Індефікатор даних в таблиці "sessions"\*/

@Id

@GeneratedValue

@Column(name = "id")

private Integer id;

/\*\*Результат сесії лікування\*/

@Column(name = "result")

@Type(type = "boolean")

private Boolean result;

/\*\*Зв'язний параметр з таблицею "cards". Відношення багато до одного\*/

@ManyToOne

@JoinColumn(name = "card\_id")

private Card card;

/\*\*Зв'язний параметр з таблицею "diagnosis". Відношення багато до одного\*/

@ManyToOne

@JoinColumn(name = "diagnosis\_id")

private Diagnos diagnos;

public Integer getId() {

return id;

}

/\*\*@return повертає результат лікування\*/

public Boolean getResult() {

return result;

}

/\*\*@return повертає об'єкт карти\*/

public Card getCard() {

return card;

}

/\*\*@return повертає об'єкт діагнозу\*/

public Diagnos getDiagnos() {

return diagnos;

}

public void setId(Integer id) {

this.id = id;

}

public void setResult(Boolean result) {

this.result = result;

}

/\*\*@param card встановлення об'єкт карти\*/

public void setCard(Card card) {

this.card = card;

}

/\*\*@param diagnos встановлення об'єкт діагнозу\*/

public void setDiagnos(Diagnos diagnos) {

this.diagnos = diagnos;

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + ((id == null) ? 0 : id.hashCode());

result = prime \* result

+ ((this.result == null) ? 0 : this.result.hashCode());

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Session other = (Session) obj;

if (id == null) {

if (other.id != null)

return false;

} else if (!id.equals(other.id))

return false;

if (result == null) {

if (other.result != null)

return false;

} else if (!result.equals(other.result))

return false;

return true;

}

}

package hibernate;

import hibernateConnect.DatabaseConnect;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.ArrayList;

import java.util.Set;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.OneToMany;

import javax.persistence.Table;

import org.hibernate.Hibernate;

import org.hibernate.Session;

import org.hibernate.annotations.GenericGenerator;

/\*\* Цей клас призначений для налаштування Hibernate для

\* коректної роботи з таблицею "users" із бази даних "hospital" \*/

@Entity

@Table(name="users")

public class User extends CommonField {

/\*\*Індефікатор даних в таблиці "users"\*/

@Id

@GeneratedValue(generator="increment")

@GenericGenerator(name="increment", strategy = "increment")

@Column(name="id")

private Integer id;

/\*\*Поле містить ім'я користувача\*/

@Column(name = "name")

private String name;

/\*\*Поле містить рівень доступу користувача\*/

@Column(name = "access\_level")

private Short access\_level;

/\*\*Поле містить телефон користувача\*/

@Column(name = "phone")

private String phone;

/\*\*Поле містить пароль користувача\*/

@Column(name = "pwd")

private String pwd;

/\*\*Опис зміної масиву даних для витягування всіх нотатків, які стосуються даної групи,

\* із зв'язаної таблиці "notes" по індефікатору "user\_id", який буде співпадати

\* з індефікатором "id" поля з таблиці "user". Відношення один до багатьох

\* \*/

@OneToMany(fetch = FetchType.LAZY, mappedBy="user", cascade = CascadeType.ALL)

private Set<Note> notes;

/\*\*@param pwd встановлює пароль. Метод шифрування MD5\*/

public void setPwd(String pwd) {

MessageDigest c = null;

try {

c = MessageDigest.getInstance("MD5");

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

}

byte[] h = c.digest(pwd.getBytes());

StringBuffer pass = new StringBuffer();

for(int i = 0; i < h.length; ++i)

pass.append(Integer.toHexString(0xFF & h[i]));

this.pwd = pass.toString();

}

/\*\*Створення об'єкта класа із параметрами

\* @param name ім'я користувача

\* @param access\_level рівень доступу користувача

\* @param phone телефон користувача

\* @param pwd пароль користувача\*/

public User( String name, Short access\_level,

String phone, String pwd) {

this.name = name;

this.access\_level = access\_level;

this.phone = phone;

this.setPwd(pwd);

}

/\*\*Створення об'єкта класа\*/

public User() {

}

public Integer getId() {

return id;

}

/\*\*@return повертає ім'я користувача\*/

public String getName() {

return name;

}

/\*\*@return повертає рівень доступу користувача\*/

public Short getAccess\_level() {

return access\_level;

}

/\*\*@return повертає телефон користувача\*/

public String getPhone() {

return phone;

}

/\*\*@return повертає пароль користувача\*/

public String getPwd() {

return pwd;

}

/\*\*@return повертає колекцію нотатків\*/

public Set<Note> getNotes() {

return notes;

}

/\*\*

\* Цей проект використовує "Lazy initialization" - це означає, що дані із зв'язаних

\* таблиць(нотатки) не завантажуються автоматично, тому в цьому класі два метода для

\* отримання даних. Перший метод "getNotes" повертає структуру даних, якщо ці дані

\* вже завантажені, другий метод "getAllNotes" створює запит до бази даних і

\* завантажує з неї необхідні дані і повертає структуру з цими даними, що

\* стосуються поточного користувача.

\* @return колекцію нотатків

\*/

public ArrayList<Note> getAllNotes() {

ArrayList<Note> listNotes;

Session session = DatabaseConnect.getSessionFactory().openSession();

User user = (User) session.load(User.class, this.id);

Hibernate.initialize(user.notes);

listNotes = new ArrayList<>(user.getNotes());

session.close();

return listNotes;

}

public void setId(Integer id) {

this.id = id;

}

/\*\*@param name встановлення ім'я користувача\*/

public void setName(String name) {

this.name = name;

}

/\*\*@param access\_level встановлення рівень доступу користувача\*/

public void setAccess\_level(Short access\_level) {

this.access\_level = access\_level;

}

/\*\*@param phone встановлення телефон користувача\*/

public void setPhone(String phone) {

this.phone = phone;

}

/\*\*@param notes встановлення колекція нотатків\*/

public void setNotes(Set<Note> notes) {

this.notes = notes;

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result

+ ((access\_level == null) ? 0 : access\_level.hashCode());

result = prime \* result + ((id == null) ? 0 : id.hashCode());

result = prime \* result + ((name == null) ? 0 : name.hashCode());

result = prime \* result + ((phone == null) ? 0 : phone.hashCode());

result = prime \* result + ((pwd == null) ? 0 : pwd.hashCode());

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

User user = (User) obj;

if(this.access\_level.equals(user.access\_level) &&

this.name.equals(user.name) &&

this.phone.equals(user.phone) &&

this.pwd.equals(user.pwd))

return true;

return false;

}

}

package hibernateConnect;

import org.hibernate.SessionFactory;

import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

import org.hibernate.cfg.Configuration;

import org.hibernate.service.ServiceRegistry;

/\*\*Клас призначиний для організації підключення до бази даних\*/

public class DatabaseConnect {

/\*\*Підключення до бази даних\*/

private static SessionFactory sessionFactory = null;

static {

Configuration config = new Configuration();

config.configure("hibernate.cfg.xml");

ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder().

applySettings(config.getProperties()).build();

sessionFactory = new Configuration().configure().

buildSessionFactory(serviceRegistry);

}

/\*\*@return повертає підключення до бази даних\*/

public static SessionFactory getSessionFactory(){

return sessionFactory;

}

}

**package** logic;

**import** hibernate.User;

/\*\*Клас призначений для авторизації користувача в системі\*/

**public** **class** AuthorizeUser {

/\*\*Авторизовани користувач\*/

**private** **static** User *authorizeUser* = **null**;

/\*\*Метод виконує процедуру авторизації

\* **@param** name ім'я користувача

\* **@param** pwd пароль користувача

\* **@return** повертає true якщо користувач пройшов авторизацію, повертає false якщо ні\*/

**public** **static** **boolean** authorizeUser(String name, String pwd) {

User user = **new** User(name, **null**, "", pwd);

**if** (((user = OperationsWithUsers.*getOperationWithUsers*().isRegisted(user)) !=

**null**) && (user.getAccess\_level() == 2)){ // User with admin rights

*authorizeUser* = user;

**return** **true**;

}

**else** **return** **false**;

}

/\*\***@return** повертає об'єкт авторизованого користувача\*/

**public** **static** User getAuthorizeUser() {

**return** *authorizeUser*;

}

/\*\***@return** повертає true якщо користувач пройшов авторизацію, інекше false\*/

**public** **static** **boolean** isAuthorizeUser() {

**if** (*authorizeUser* == **null**)

**return** **false**;

**else** **return** **true**;

}

/\*\*Метод встановлює, що користувач вийшов із сиситеми\*/

**public** **static** **void** setUserLogOut() {

*authorizeUser* = **null**;

}

}

package logic;

import hibernate.CommonField;

import hibernateConnect.DatabaseConnect;

import java.util.ArrayList;

import java.util.List;

import org.hibernate.Criteria;

import org.hibernate.HibernateException;

import org.hibernate.Session;

import org.hibernate.Transaction;

import exceptions.ProjectException;

/\*\*Абстрактний клас призначиний для реалізації загальних операцій з даними із бази даних\*/

abstract class CommonOperations<I extends CommonField, Filtr> {

/\*\*Абстрактний метод, який має реалізувати фільтр

\* @param session сесія з базою даних

\* @param filtr параметри для фільтрації даних з бази даних

\* @return запит до бази даних для витягування всіх об'єктів, які задовільняють фільтр\*/

abstract Criteria getFiltrCriteria(Session session, Filtr filtr);

/\*\*Абстрактний метод, який має реалізувати запит до бази даних

\* @param session сесія з базою даних

\* @return запит до бази даних для витягування всіх об'єктів класу "I"\*/

abstract Criteria getAllObjCriteria(Session session);

/\*\*Абстрактний метод, який реалізує критерії, за якими шукається дані в таблиці

\* @param session сесія з базою даних

\* @param object об'єкт, який шукається в базі даних

\* @return запит до бази даних для витягування всіх об'єктів, які задовільняють критерії\*/

abstract Criteria getRegisterCriteria(Session session, I object);

/\*\*Абстрактний метод, який реалізує використання методу getRegisterCriteria

\* @return повертати true потрібно, коли використання метода getRegisterCriteria не потрібно

\* (збільшує продуктивність), при повернені false метод getRegisterCriteria

\* буде використовуватися

\*/

abstract boolean isRegistedMethodNotNeed();

/\*\*Метод видаляє із бази даних всі дані пов'язанні з object, для пошука

\* індефікатора використовується getRegisterCriteria

\* @param object дані, які треба видалити

\* @return true якщо видалення пройшло успішно і false якщо ні\*/

public boolean delete(I object) {

if ((object = isRegisted(object)) == null)

return false;

else {

Session session = null;

try {

session = DatabaseConnect.getSessionFactory().openSession();

session.beginTransaction();

session.delete(object);

session.getTransaction().commit();

} catch (Exception e) {

ProjectException.showMessage("Delete " + object.toString(), e);

return false;

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return true;

}

}

/\*\*Видаляє дані по їх індефікатору і всі інші дані пов'язані з ним.

\* Не використовує getRegisterCriteria

\* @param object дані для видалення

\* @return true якщо видалення пройшло успішно і false якщо ні\*/

public boolean deleteById(I object) {

Session session = null;

try {

session = DatabaseConnect.getSessionFactory().openSession();

session.beginTransaction();

session.delete(object);

session.getTransaction().commit();

} catch (Exception e) {

ProjectException.showMessage("Delete " + object.toString(), e);

return false;

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return true;

}

/\*\*Метод витягує всі дані з таблиці, які задоволбняють фільтр

\* @param filtr параметри фільтра

\* @return повертає колекцію об'єктів, що задовольняють фільтр\*/

@SuppressWarnings("unchecked")

protected ArrayList<I> getAllObjSatisfyFiltr(Filtr filtr) {

Session session = null;

ArrayList<I> users = null;

try {

session = DatabaseConnect.getSessionFactory().openSession();

users = new ArrayList<>(getFiltrCriteria(session, filtr).list());

} catch (Exception e) {

ProjectException.showMessage("AllObjSatisfyFiltr " + filtr.toString(), e);

return users;

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return users;

}

/\*\*Метод для регістрації об'єкта

\* @param object об'єкт для збереження в базі даних

\* @return повертає true якщо об'єкт успішно збережень у базу даних, false якщо ні\*/

public boolean register(I object) {

Session session = null;

try {

session = DatabaseConnect.getSessionFactory().openSession();

if ( isRegistedMethodNotNeed() || isRegisted(object) == null) {

Transaction transaction = session.beginTransaction();

session.save(object);

transaction.commit();

}

else return false;

} catch (HibernateException e) {

ProjectException.showMessage("Register " + object.toString(), e);

return false;

} finally {

if(session != null && session.isOpen()) {

session.close();

}

}

return true;

}

/\*\*Перевіряє чи об'єкт вже є в базі даних для цього використовує

\* пошук getRegisterCriteria

\* @param object об'єкт для перевірки

\* @return повертає цей об'єкт із бази даних або null якщо таких даних немає\*/

public I isRegisted(I object) {

if ( !isRegistedMethodNotNeed()) {

Session session = null;

try {

session = DatabaseConnect.getSessionFactory().openSession();

@SuppressWarnings("unchecked")

List<I> objects = getRegisterCriteria(session, object).list();

switch (objects.size()) {

case 0 : return null;

case 1 : object = objects.get(0); break;

default : throw new ProjectException(String.valueOf(objects.size()) +

" identical records");

}

} catch (HibernateException e) {

ProjectException.showMessage("IsRegisted Hibernet " + object.toString(), e);

return null;

} catch (ProjectException e) {

e.showMessage();

return null;

} finally {

if(session != null && session.isOpen()) {

session.close();

}

}

}

return object;

}

/\*\*Метод обновлює дані в базі даних

\* @param object дані, які необхідно оновити

\* @return true якщо оновлення виконано успішно, false якщо ні\*/

public boolean update(I object) {

I curObject = object;

if ((curObject = isRegisted(curObject)) == null)

return false;

else {

Session session = null;

object.setId(curObject.getId());

try {

session = DatabaseConnect.getSessionFactory().openSession();

session.beginTransaction();

session.update(object);

session.getTransaction().commit();

} catch (Exception e) {

ProjectException.showMessage("Ubdate " + object.toString(), e);

return false;

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return true;

}

}

/\*\*Метод повертає всі дані з таблиці

\* @return повертає колекцію всіх даних із таблиці\*/

@SuppressWarnings("unchecked")

public ArrayList<I> getAllObj() {

Session session = null;

ArrayList<I> users = null;

try {

session = DatabaseConnect.getSessionFactory().openSession();

users = new ArrayList<>(getAllObjCriteria(session).list());

} catch (Exception e) {

ProjectException.showMessage("AllObj ", e);

return users;

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return users;

}

/\*\*Видаляє всі дані з таблиці один за одгим\*/

public void deleteAllObj() {

for(I obj: getAllObj())

delete(obj);

}

}

package logic;

import java.util.ArrayList;

import hibernate.Card;

import hibernate.Gender;

import logic.OperationsWithCards.Filtr;

import org.hibernate.Criteria;

import org.hibernate.Session;

import org.hibernate.criterion.Order;

import org.hibernate.criterion.Restrictions;

public class OperationsWithCards extends CommonOperations<Card, Filtr> {

/\*\*Поле призначене для зберігання силки на об'єкт цього класу\*/

private static OperationsWithCards oper = null;

/\*\*Метод створюэ об'єкт цього класу\*/

private OperationsWithCards() {}

/\*\*Метод повертає об'єкт цього класу

\* @return повертає об'єкт класу\*/

public static OperationsWithCards getOperationWithCard() {

if(oper == null)

oper = new OperationsWithCards();

return oper;

}

/\*\*Клас, що описує поля таблиці, які будуть використовуватися для фільтру\*/

class Filtr{

private String name;

private Short age\_low;

private Short age\_hight;

private Gender sex;

private Boolean isAgain;

/\*\*Створення об'єкта класа із заданням полів

\* @param name ім'я пацієнта цієї картки

\* @param age\_levelLow нижчий вік пацієнта цієї картки

\* @param age\_levelHight вищий вік пацієнта цієї картки

\* @param gender стать пацієнта цієї картки

\* @param isAgain чи вперше пацієнт

\* \*/

Filtr(String name,

Gender gender, Short age\_levelLow, Short age\_levelHight, Boolean isAgain) {

this.name = name;

this.sex = gender;

this.age\_low = age\_levelLow;

this.age\_hight = age\_levelHight;

this.isAgain = isAgain;

}

}

@Override

Criteria getRegisterCriteria(Session session, Card card) {

if (card.getId() != null)

return session.createCriteria(Card.class).add(

Restrictions.eqOrIsNull("id", card.getId()));

else return session.createCriteria(Card.class).add(

Restrictions.eq("name", card.getName())).add(

Restrictions.eq("age", card.getAge())).add(

Restrictions.eq("sex", card.getSex())).add(

Restrictions.eq("isAgain", card.getIsAgain()));

}

@Override

Criteria getFiltrCriteria(Session session, Filtr filtr) {

Criteria criteria = session.createCriteria(Card.class).

add(Restrictions.like("name", "%" + filtr.name + "%"));

if (filtr.age\_hight != null && filtr.age\_low != null)

criteria.add(Restrictions.between("age", filtr.age\_low, filtr.age\_hight));

if (filtr.sex != null)

criteria = criteria.add(Restrictions.eq("sex", filtr.sex));

if (filtr.isAgain != null)

criteria = criteria.add(Restrictions.eq("isAgain", filtr.isAgain));

return criteria.addOrder(Order.asc("name"));

}

@Override

Criteria getAllObjCriteria(Session session) {

return session.createCriteria(Card.class);

}

/\*\*Метод для отримання всіх даних з таблиці, що задовільняєть фільтр {@link getFiltrCriteria}

\* @param name задання ім'я хворого

\* @param gender задання статі хворого

\* @param age\_levelLow задання нижчого порогу віку

\* @param age\_levelHight задання вищого порогу віку

\* @param isAgain задання чи хворий не вперше

\* @return колекцію карт\*/

public ArrayList<Card> getAllCardsFiltr(String name,

Gender gender, Short age\_levelLow, Short age\_levelHight, Boolean isAgain) {

return getAllObjSatisfyFiltr(new Filtr(name, gender, age\_levelLow,

age\_levelHight, isAgain));

}

@Override

boolean isRegistedMethodNotNeed() {

return false;

}

}

package logic;

import org.hibernate.Criteria;

import org.hibernate.Session;

import hibernate.Diagnos;

import logic.OperationsWithDiagnosis.Filtr;

public class OperationsWithDiagnosis extends CommonOperations<Diagnos, Filtr> {

/\*\*Поле призначене для зберігання силки на об'єкт цього класу\*/

private static OperationsWithDiagnosis oper = null;

/\*\*Метод створюэ об'єкт цього класу\*/

private OperationsWithDiagnosis() {}

/\*\*Метод повертає об'єкт цього класу

\* @return повертає об'єкт класу\*/

public static OperationsWithDiagnosis getOperationsWithDiagnosis() {

if (oper == null)

oper = new OperationsWithDiagnosis();

return oper;

}

/\*\*Клас, що описує поля таблиці, які будуть використовуватися для фільтру\*/

class Filtr {

// Фільтр не потрібен

}

@Override

Criteria getFiltrCriteria(Session session, Filtr filtr) {

// TODO Auto-generated method stub

return null;

}

@Override

Criteria getAllObjCriteria(Session session) {

return session.createCriteria(Diagnos.class);

}

@Override

Criteria getRegisterCriteria(Session session, Diagnos object) {

// TODO Auto-generated method stub

return null;

}

@Override

boolean isRegistedMethodNotNeed() {

// TODO Auto-generated method stub

return true;

}

}

package logic;

import org.hibernate.Criteria;

import org.hibernate.Session;

import org.hibernate.criterion.Order;

import org.hibernate.criterion.Restrictions;

import hibernate.Disease;

import logic.OperationsWithDiseases.Filtr;

public class OperationsWithDiseases extends CommonOperations<Disease, Filtr> {

/\*\*Клас, що описує поля таблиці, які будуть використовуватися для фільтру\*/

class Filtr {

private String name;

/\*\*Створення об'єкта класа із параметрами

\* @param name ім'я хвороби\*/

Filtr(String name) {

this.name = name;

}

}

/\*\*Поле призначене для зберігання силки на об'єкт цього класу\*/

private static OperationsWithDiseases oper = null;

/\*\*Метод створюэ об'єкт цього класу\*/

private OperationsWithDiseases() {}

/\*\*Метод повертає об'єкт цього класу

\* @return повертає об'єкт класу\*/

public static OperationsWithDiseases getOperationsWithDiseases() {

if (oper == null)

oper = new OperationsWithDiseases();

return oper;

}

@Override

Criteria getFiltrCriteria(Session session, Filtr filtr) {

return session.createCriteria(Disease.class).

add(Restrictions.like("name", "%" + filtr.name +

"%")).addOrder(Order.asc("name"));

}

@Override

Criteria getAllObjCriteria(Session session) {

return session.createCriteria(Disease.class);

}

@Override

Criteria getRegisterCriteria(Session session, Disease object) {

if (object.getId() != null)

return session.createCriteria(Disease.class).add(Restrictions.eq("id", object.getId()));

else return session.createCriteria(Disease.class).add(Restrictions.like("name",

object.getName()));

}

@Override

boolean isRegistedMethodNotNeed() {

return false;

}

}

package logic;

import hibernate.Group;

import java.util.ArrayList;

import logic.OperationsWithGroups.Filtr;

import org.hibernate.Criteria;

import org.hibernate.Session;

import org.hibernate.criterion.Restrictions;

public class OperationsWithGroups extends CommonOperations<Group, Filtr>{

/\*\*Поле призначене для зберігання силки на об'єкт цього класу\*/

private static OperationsWithGroups oper = null;

/\*\*Метод створюэ об'єкт цього класу\*/

private OperationsWithGroups() {}

/\*\*Метод повертає об'єкт цього класу

\* @return повертає об'єкт класу\*/

public static OperationsWithGroups getOperationsWithGroups() {

if (oper == null)

oper = new OperationsWithGroups();

return oper;

}

/\*\*Клас, що описує поля таблиці, які будуть використовуватися для фільтру\*/

class Filtr {

private String name;

/\*\*Створення об'єкта класа із параметрами

\* @param name ім'я групи\*/

Filtr(String name) {

this.name = name;

}

}

// Повертає лише голови груп

@Override

Criteria getFiltrCriteria(Session session, Filtr filtr) {

return session.createCriteria(Group.class).add(Restrictions.like("name", "%" +

filtr.name + "%")).add(Restrictions.eq("isHead", true));

}

@Override

Criteria getAllObjCriteria(Session session) {

return session.createCriteria(Group.class).add(Restrictions.eq("isHead", true));

}

@Override

Criteria getRegisterCriteria(Session session, Group object) {

return session.createCriteria(Group.class).add(Restrictions.eq("name",

object.getName()));

}

@Override

boolean isRegistedMethodNotNeed() {

return false;

}

/\*\*Метод для отримання всіх даних з таблиці, що задовільняєть фільтр {@link getFiltrCriteria}

\* @param name задання ім'я групи

\* @return колекцію груп\*/

public ArrayList<Group> getAllObjSatisfyFiltr(String name) {

return getAllObjSatisfyFiltr(new Filtr(name));

}

}

package logic;

import hibernate.User;

import java.util.ArrayList;

import logic.OperationsWithUsers.Filtr;

import org.hibernate.Criteria;

import org.hibernate.Session;

import org.hibernate.criterion.Order;

import org.hibernate.criterion.Restrictions;

public class OperationsWithUsers extends CommonOperations<User, Filtr> {

/\*\*Поле призначене для зберігання силки на об'єкт цього класу\*/

private static OperationsWithUsers operWithUsers = null;

/\*\*Метод створюэ об'єкт цього класу\*/

private OperationsWithUsers() {}

/\*\*Метод повертає об'єкт цього класу

\* @return повертає об'єкт класу\*/

public static OperationsWithUsers getOperationWithUsers() {

if (operWithUsers == null)

operWithUsers = new OperationsWithUsers();

return operWithUsers;

}

/\*\*Клас, що описує поля таблиці, які будуть використовуватися для фільтру\*/

class Filtr {

private String name;

private String phone;

private Short access\_levelLow;

private Short access\_levelHight;

/\*\*Створення об'єкта класа із параметрами

\* @param name ім'я користувача

\* @param access\_levelLow нижчий рівень доступу користувача

\* @param access\_levelHight вищий рівень доступу користувача

\* @param phone телефон користувача

\* @param pwd пароль користувача\*/

Filtr(String name, String pwd,

String phone, Short access\_levelLow, Short access\_levelHight) {

this.name = name;

this.phone = phone;

this.access\_levelLow = access\_levelLow;

this.access\_levelHight = access\_levelHight;

}

}

@Override

Criteria getRegisterCriteria(Session session, User user) {

if (user.getId() == null)

return session.createCriteria(User.class).add(

Restrictions.eq("name", user.getName())).add(

Restrictions.eq("pwd", user.getPwd()));

else return session.createCriteria(User.class).add(

Restrictions.eq("id", user.getId()));

}

@Override

Criteria getFiltrCriteria(Session session, Filtr filtr) {

Criteria criteria = session.createCriteria(User.class).add(Restrictions.

like("name", "%" + filtr.name + "%")).add(Restrictions.like("phone",

"%" + filtr.phone + "%"));

if (filtr.access\_levelHight != null && filtr.access\_levelLow != null)

criteria.add(Restrictions.between("access\_level",

filtr.access\_levelLow, filtr.access\_levelHight));

return criteria.addOrder(Order.asc("name"));

}

@Override

Criteria getAllObjCriteria(Session session) {

return session.createCriteria(User.class);

}

/\*\*Метод для отримання всіх даних з таблиці, що задовільняєть фільтр {@link getFiltrCriteria}

\* @param name задання ім'я користувача

\* @param phone задання телефону користувача

\* @param access\_levelLow задання нижчого порогу доступу

\* @param access\_levelHight задання вищого порогу доступу

\* @return колекцію користувачів\*/

public ArrayList<User> getAllObjSatisfyFiltr(String name,

String phone, Short access\_levelLow, Short access\_levelHight) {

return getAllObjSatisfyFiltr(new Filtr(name, null, phone, access\_levelLow,

access\_levelHight));

}

@Override

boolean isRegistedMethodNotNeed() {

return false;

}

}

package logic;

import hibernate.Note;

import java.sql.Date;

import java.util.ArrayList;

import logic.OperationWithNotes.Filtr;

import org.hibernate.Criteria;

import org.hibernate.Session;

import org.hibernate.criterion.Restrictions;

public class OperationWithNotes extends CommonOperations<Note, Filtr> {

/\*\*Поле призначене для зберігання силки на об'єкт цього класу\*/

private static OperationWithNotes oper = null;

/\*\*Метод створюэ об'єкт цього класу\*/

private OperationWithNotes() {}

/\*\*Метод повертає об'єкт цього класу

\* @return повертає об'єкт класу\*/

public static OperationWithNotes getOperationWithNotes() {

if (oper == null)

oper = new OperationWithNotes();

return oper;

}

/\*\*Клас, що описує поля таблиці, які будуть використовуватися для фільтру\*/

class Filtr {

private Boolean hide;

private Date date\_low;

private Date date\_hight;

/\*\*Створення об'єкта класа із параметрами

\* @param hide режим видимості

\* @param date\_low задає нижчий рівень час

\* @param date\_hight задає вищий рівень час\*/

Filtr(Boolean hide, Date date\_low, Date date\_hight) {

this.hide = hide;

this.date\_hight = date\_hight;

this.date\_low = date\_low;

}

}

@Override

Criteria getRegisterCriteria(Session session, Note object) {

return null;

}

@Override

Criteria getFiltrCriteria(Session session, Filtr filtr) {

Criteria criteris = session.createCriteria(Note.class);

if ( filtr.date\_low != null && filtr.date\_hight != null)

criteris.add(Restrictions.between("date", filtr.date\_low, filtr.date\_hight));

if (filtr.hide != null)

criteris.add(Restrictions.eq("hide", filtr.hide));

return criteris;

}

@Override

Criteria getAllObjCriteria(Session session) {

return session.createCriteria(Note.class);

}

@Override

boolean isRegistedMethodNotNeed() {

return true;

}

/\*\*Метод для отримання всіх даних з таблиці, що задовільняєть фільтр {@link getFiltrCriteria}

\* @param hide задання стану нотатку

\* @param date\_low задання нижчого порогу часу

\* @param date\_hight задання вищого порогу часу

\* @return колекцію нотатків\*/

public ArrayList<Note> getAllObjSatisfyFiltr(Boolean hide, Date date\_low,

Date date\_hight) {

return getAllObjSatisfyFiltr(new Filtr(hide, date\_low, date\_hight));

}

}

package logic;

import hibernate.Session;

import java.util.ArrayList;

import logic.OperationWithSessions.Filtr;

import org.hibernate.Criteria;

import org.hibernate.criterion.Restrictions;

public class OperationWithSessions extends CommonOperations<Session, Filtr> {

/\*\*Поле призначене для зберігання силки на об'єкт цього класу\*/

private static OperationWithSessions oper = null;

/\*\*Метод створюэ об'єкт цього класу\*/

private OperationWithSessions() {}

/\*\*Метод повертає об'єкт цього класу

\* @return повертає об'єкт класу\*/

public static OperationWithSessions getOperationWithSessions() {

if (oper == null)

oper = new OperationWithSessions();

return oper;

}

/\*\*Клас, що описує поля таблиці, які будуть використовуватися для фільтру\*/

class Filtr {

private Boolean result;

/\*\*Створення об'єкта класа із параметром

\* @param result задає результат лікування\*/

Filtr(Boolean result) {

this.result = result;

}

}

@Override

Criteria getFiltrCriteria(org.hibernate.Session session, Filtr filtr) {

Criteria criteris = session.createCriteria(Session.class);

if (filtr.result != null)

criteris.add(Restrictions.eq("result", filtr.result));

return null;

}

@Override

Criteria getAllObjCriteria(org.hibernate.Session session) {

return session.createCriteria(Session.class);

}

@Override

Criteria getRegisterCriteria(org.hibernate.Session session, Session object) {

return null;

}

@Override

boolean isRegistedMethodNotNeed() {

return true;

}

/\*\*Метод для отримання всіх даних з таблиці, що задовільняєть фільтр {@link getFiltrCriteria}

\* @param result задання результату лікування

\* @return колекцію сесій\*/

public ArrayList<Session> getAllObjSatisfyFiltr(Boolean result) {

return getAllObjSatisfyFiltr(new Filtr(result));

}

}

package pagesServlets;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import logic.AuthorizeUser;

/\*\*Клас-сервелет для авторизації користувача\*/

public class AuthorizeServlet extends HttpServlet{

/\*\*Унікальний індефікатор для сервелету\*/

private static final long serialVersionUID = 1L;

/\*\*Метод приймає введені дані на формі користувачем і викликає

\* метод AuthorizeUser.authorizeUser для йогот авторизації

\* @param request запит, який приходить від користувача

\* @param response відповідь, яка надсилається користувачу\*/

@Override

protected void doPost(HttpServletRequest request,

HttpServletResponse response) throws ServletException, IOException {

request.setCharacterEncoding("utf8");

if (AuthorizeUser.authorizeUser(request.getParameter("userName"),

request.getParameter("password")))

response.sendRedirect("StartPage.jsp");

else {

HttpSession session = request.getSession();

session.setAttribute("ErrorLogIn", "TRUE");

response.sendRedirect("Authorize.jsp");

}

}

}

package pagesServlets;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import logic.AuthorizeUser;

/\*\*Клас-сервелет реалізовує вихід користувача\*/

public class LogOutServlet extends HttpServlet {

/\*\*Унікальний індефікатор для сервелету\*/

private static final long serialVersionUID = 361381042632159371L;

/\*\*Метод виконує операцію виходу користувача і викликає

\* метод AuthorizeUser.setUserLogOut для йогот виходу із системи

\* @param request запит, який приходить від користувача

\* @param response відповідь, яка надсилається користувачу\*/

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

request.setCharacterEncoding("utf8");

HttpSession session = request.getSession();

if (session != null)

session.invalidate();

AuthorizeUser.setUserLogOut();

response.sendRedirect("Authorize.jsp");

}

}

package pagesServlets;

import java.io.IOException;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import logic.AuthorizeUser;

/\*\*Клас-фільтр для захисту від не авторизованого доступу до системи\*/

public class MyFilter implements Filter {

/\*\*Метод викликається при знищені об'єкта фільтра\*/

@Override

public void destroy() {

}

/\*\*Метод викликається при будь-яких запитах і відповідях клієнту.

\* Метод перевіряє чи користувач авторизован, якщо так то передати

\* далі запит\відповідь, якщо ні перенаправити користувача на

\* сторіну авторизації

\* @param request запит клієнта

\* @param response відповідь клієнта

\* @param chain ланцюг фільтрів\*/

@Override

public void doFilter(ServletRequest request, ServletResponse response,

FilterChain chain) throws IOException, ServletException {

request.setCharacterEncoding("utf8");

HttpServletRequest req = (HttpServletRequest) request;

HttpServletResponse res = (HttpServletResponse) response;

String uri = req.getRequestURI();

if (!AuthorizeUser.isAuthorizeUser() && !(uri.endsWith("LogIn") ||

uri.endsWith("Authorize.jsp") ||

uri.endsWith("/BD\_Electronic\_System\_Hospital\_/")))

{

res.sendRedirect("Authorize.jsp");

} else

chain.doFilter(request, response);

}

/\*\*Метод викликається при створені об'єкта фільтра

\* @param fConfig налаштування для фільтра\*/

@Override

public void init(FilterConfig fConfig) throws ServletException {

}

}

package tests;

import static logic.OperationsWithCards.getOperationWithCard;

import static logic.OperationsWithUsers.getOperationWithUsers;

import hibernate.Card;

import hibernate.Gender;

import hibernate.User;

/\*\*Тестовий клас для тесту працездатності проекту\*/

public class Test {

/\*\*Метод наповнює базу даних\*/

public static void main(String[] args) {

Tests testRelat = new Tests();

testRelat.cleanAllTables();

testRelat.testRelationship();

getOperationWithUsers().register(new User("Адміністратор Петрик Васильович", (short) 2, "+38095230294", "0000"));

getOperationWithUsers().register(new User("Смерека Вадим Петрович", (short) 2, "+38095345961", "1111"));

getOperationWithUsers().register(new User("Каратишко Даша Вікторівна", (short) 0, "+38092241526", "2222"));

getOperationWithUsers().register(new User("Сонько Петро Васильовіч", (short) 1, "+38034536367", "3333"));

getOperationWithUsers().register(new User("Ціпко Маша Григорівна", (short) 0, "+380889214546", "4444"));

for(int i = 0;i < 7; i++)

getOperationWithCard().register(new Card("Карта " + String.valueOf(i), (short) (30 + i),

i % 2 == 0 ? Gender.female : Gender.male, false));

}

}

package tests;

import static logic.OperationsWithCards.getOperationWithCard;

import static logic.OperationsWithUsers.getOperationWithUsers;

import static org.junit.Assert.assertTrue;

import static org.junit.Assert.fail;

import hibernate.Card;

import hibernate.Diagnos;

import hibernate.Disease;

import hibernate.Gender;

import hibernate.Note;

import hibernate.User;

import hibernateConnect.DatabaseConnect;

import java.sql.Date;

import java.util.ArrayList;

import java.util.Comparator;

import java.util.List;

import org.hibernate.Session;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

/\*\*Клас для точочних тестів функціоналу проекту\*/

public class Tests {

ArrayList<User> users;

ArrayList<Card> cards;

/\*\*Метод створює об'єкт тестування і задає початкові налаштування\*/

public Tests() {

users = new ArrayList<>();

users.add(new User("Porky1",(short) 0, "134321223", "1111"));

users.add(new User("Porky2",(short) 1, "132135223", "0000"));

users.add(new User("Porky3",(short) 2, "134633432", "jsfn"));

users.add(new User("Porky4",(short) 2, "435446333", "sdfg"));

users.add(new User("Porky5",(short) 1, "123456789", "dsgd"));

users.add(new User("Porky6",(short) 0, "444325252", "dsfd"));

users.add(new User("Porky7",(short) 0, "657345434", "dsfs"));

cards = new ArrayList<>();

for(int i = 0;i < 7; i++)

cards.add(new Card("Card" + String.valueOf(i), (short) (30 + i),

i % 2 == 0 ? Gender.female : Gender.male, false));

}

/\*\*Тест на регістрацію карток\*/

@Test

public void testRegisterCard() {

getOperationWithUsers().deleteAllObj();

for(int i = 0; i < users.size(); i++)

if(!getOperationWithUsers().register(users.get(i)))

fail();

for(int i = 0; i < users.size(); i++)

if(getOperationWithUsers().isRegisted(users.get(i)) == null)

fail();

}

/\*\*Тест на видалення користувачів\*/

@Test

public void testDeleteUsers() {

for(int i = 0; i < users.size(); i++)

getOperationWithUsers().register(users.get(i));

for(int i = 0; i < users.size(); i++)

if(!getOperationWithUsers().delete(users.get(i)))

fail();

}

/\*\*Тест на фільтр користувачів\*/

@Test

public void testUsersFiltr() {

getOperationWithUsers().deleteAllObj();

for(int i = 0; i < users.size(); i++)

if(!getOperationWithUsers().register(users.get(i)))

fail();

if(getOperationWithUsers().getAllObjSatisfyFiltr("Porky",

"",(short) 1,(short) 1).size() != 2)

fail();

if(getOperationWithUsers().getAllObjSatisfyFiltr("or",

"132135223",(short) 0,(short) 2).size() != 1)

fail();

if(getOperationWithUsers().getAllObjSatisfyFiltr("ork",

"",(short) 0,(short) 2).size() != 7)

fail();

}

/\*\*Тест на оновлення даних користувачів\*/

@Test

public void testUpdateUser() {

getOperationWithUsers().deleteAllObj();

for(int i = 0; i < users.size(); i++)

if(!getOperationWithUsers().register(users.get(i)))

fail();

for(int i = 0; i < users.size(); i++)

if(!getOperationWithUsers().update(users.get(i)))

fail();

}

/\*\*Тест на витягувування даних з бази даних\*/

@Test

public void testGetUsers() {

getOperationWithUsers().deleteAllObj();

for(int i = 0; i < users.size(); i++)

if(!getOperationWithUsers().register(users.get(i)))

fail();

List<User> usersP = getOperationWithUsers().getAllObj();

for(int i = 0; i < users.size(); ++i)

assertTrue(usersP.get(i).equals(users.get(i)));

}

/\*\*Тест на регістрацію карток\*/

@Test

public void testRegisteCards() {

getOperationWithCard().deleteAllObj();

for(int i = 0; i < users.size(); i++)

if(!getOperationWithCard().register(cards.get(i)))

fail();

for(int i = 0; i < users.size(); i++)

if(getOperationWithCard().isRegisted(cards.get(i)) == null)

fail();

}

/\*\*Тест на видалити картки\*/

@Test

public void testDeleteCards() {

for(int i = 0; i < cards.size(); i++)

getOperationWithCard().register(cards.get(i));

for(int i = 0; i < cards.size(); i++)

if(!getOperationWithCard().delete(cards.get(i)))

fail();

}

/\*\*Тест на використання фільтра карток\*/

@Test

public void testCardFiltr() {

getOperationWithCard().deleteAllObj();

for(int i = 0; i < cards.size(); i++)

if(!getOperationWithCard().register(cards.get(i)))

fail();

if(getOperationWithCard().getAllCardsFiltr("Card", null,(short)

0,(short) 40, null).size() != 7)

fail();

if(getOperationWithCard().getAllCardsFiltr("ard",

Gender.female,(short) 0,(short) 40, null).size() != 4)

fail();

if(getOperationWithCard().getAllCardsFiltr("ork",

null,(short) 0,(short) 35, null).size() != 0)

fail();

}

/\*\*Тест на оновлення карток\*/

@Test

public void testUpdateCard() {

getOperationWithCard().deleteAllObj();

for(int i = 0; i < cards.size(); i++)

if(!getOperationWithCard().register(cards.get(i)))

fail();

for(int i = 0; i < cards.size(); i++)

if(!getOperationWithCard().update(cards.get(i)))

fail();

}

/\*\*Тест на витягування карток із бази даних\*/

@Test

public void testGetCards() {

getOperationWithCard().deleteAllObj();

for(int i = 0; i < cards.size(); i++)

if(!getOperationWithCard().register(cards.get(i)))

fail();

List<Card> cardsP = getOperationWithCard().getAllObj();

for(int i = 0; i < cards.size(); ++i)

assertTrue(cardsP.get(i).equals(cards.get(i)));

}

/\*\*Комплексний тест на всі таблиці(тестування зв'язку між таблицями\*/

// Я знаю виглядає жахливо.

@Test

public void testRelationship() {

Session session = DatabaseConnect.getSessionFactory().openSession();

session.beginTransaction();

int size = 2;

User[] user = new User[size];

user[0] = new User("Свиненко Петро Висильович",(short) 0, "+380314567849", "1234");

user[1] = new User("Хабрович Вадим Петрович",(short) 0, "+38076589165", "4321");

Note[] note = new Note[size];

note[0] = new Note(false, Date.valueOf("2000-09-01"), "Хворий скаржиться на біль у голові");

note[1] = new Note(true, Date.valueOf("2012-11-04"), "Хворий має проблеми із орінтацією");

Card card = new Card("Бочко Андій Вікторович", (short) 28, Gender.male, true);

card.setNote("Хворий досить дивний. Будьте уважні.");

hibernate.Session[] sessions = new hibernate.Session[size];

sessions[0] = new hibernate.Session();

sessions[1] = new hibernate.Session();

sessions[0].setResult(true);

sessions[1].setResult(false);

Diagnos[] diagnos = new Diagnos[2];

diagnos[0] = new Diagnos("У хворого відсутній мозок");

diagnos[1] = new Diagnos("У хворого стався інцест. Мозок знову відсутній");

Disease[] disease = new Disease[2];

disease[0] = new Disease("Немає мозку. Немає головного мозку");

disease[1] = new Disease("Немає мозку. Немає довгого мозку");

for(int i = 0; i < size; ++i) {

diagnos[i].setDisease(disease[i]);

sessions[i].setDiagnos(diagnos[i]);

sessions[i].setCard(card);

note[i].setCard(card);

note[i].setUser(user[i]);

}

session.save(card);

for(int i = 0; i < size; ++i) {

session.save(user[i]);

session.save(note[i]);

session.save(disease[i]);

session.save(diagnos[i]);

session.save(sessions[i]);

}

session.getTransaction().commit();

session.close();

List<Card> cardsD = getOperationWithCard().getAllObj();

Card cardD = cardsD.get(0);

ArrayList<Note> notesD = cardD.getAllNotes();

notesD.sort(new Comparator<Note>() {

@Override

public int compare(Note arg0, Note arg1) {

if(arg0.getId() > arg1.getId())

return 1;

else return -1;

}

});

ArrayList<hibernate.Session> sessionsD = cardD.getAllSessions();

sessionsD.sort(new Comparator<hibernate.Session>() {

@Override

public int compare(hibernate.Session arg0, hibernate.Session arg1) {

if(arg0.getId() > arg1.getId())

return 1;

else return -1;

}

});

for(int i = 0; i < size; ++i) {

Note noteD = notesD.get(i);

User userD = noteD.getUser();

hibernate.Session sessionD = sessionsD.get(i);

Diagnos diagnosD = sessionD.getDiagnos();

Disease diseaseD = diagnosD.getDisease();

assertTrue(user[i].equals((Object) userD));

assertTrue(note[i].equals(noteD));

assertTrue(card.equals(cardD));

assertTrue(sessions[i].equals(sessionD));

assertTrue(diagnos[i].equals(diagnosD));

assertTrue(disease[i].equals(diseaseD));

}

}

/\*\*Метод очищає базу даних перед ткстом і після\*/

@Before

@After

public void cleanAllTables() {

Session session = DatabaseConnect.getSessionFactory().openSession();

session.beginTransaction();

String tablesName[] = {"cards", "diagnosis", "diseases", "groups", "notes",

"sessions", "users"};

session.createSQLQuery("SET FOREIGN\_KEY\_CHECKS = 0").executeUpdate();

for(int i = 0; i < tablesName.length; ++i) {

session.createSQLQuery("TRUNCATE " + tablesName[i]).executeUpdate();

}

session.createSQLQuery("SET FOREIGN\_KEY\_CHECKS = 1").executeUpdate();

session.getTransaction().commit();

session.close();

}

}

Hibernate.cfg.xml

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<property name=*"hibernate.connection.characterEncoding"*>utf8</property>

<property name=*"hibernate.connection.url"*>jdbc:mysql://localhost:3306/hospital</property>

<property name=*"hibernate.connection.username"*>root</property>

<property name=*"hibernate.connection.password"*>1111</property>

<property name=*"connection.pool\_size"*>1</property>

<property name=*"dialect"*>org.hibernate.dialect.MySQLDialect</property>

<property name=*"current\_session\_context\_class"*>thread</property>

<property name=*"cache.provider\_class"*>org.hibernate.cache.NoCacheProvider</property>

<property name=*"show\_sql"*>true</property>

<mapping class=*"hibernate.User"*/>

<mapping class=*"hibernate.Card"*/>

<mapping class=*"hibernate.Note"*/>

<mapping class=*"hibernate.Session"*/>

<mapping class=*"hibernate.Diagnos"*/>

<mapping class=*"hibernate.Group"*/>

<mapping class=*"hibernate.Disease"*/>

</session-factory>

</hibernate-configuration>

Web.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* id=*"WebApp\_ID"* metadata-complete=*"true"* version=*"2.5"*>

<display-name>BD\_Electronic\_System\_Hospital\_</display-name>

<display-name>HibernateWebApp</display-name>

<servlet>

<display-name>User</display-name>

<servlet-name>User</servlet-name>

<servlet-class>pagesServlets.AuthorizeServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>User</servlet-name>

<url-pattern>/LogIn</url-pattern>

</servlet-mapping>

<servlet>

<display-name>Log out from acount</display-name>

<servlet-name>LogOut</servlet-name>

<servlet-class>pagesServlets.LogOutServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>LogOut</servlet-name>

<url-pattern>/LogOutServlet</url-pattern>

</servlet-mapping>

<servlet>

<display-name>Modify user data</display-name>

<servlet-name>ModifyUserData</servlet-name>

<servlet-class>pagesServlets.ModifyUserDataServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>ModifyUserData</servlet-name>

<url-pattern>/ModifyUser</url-pattern>

</servlet-mapping>

<filter>

<filter-name>myFilter</filter-name>

<filter-class>pagesServlets.MyFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>myFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<welcome-file-list>

<welcome-file>Authorize.jsp</welcome-file>

</welcome-file-list>

</web-app>

Authorize.jsp

<%@page import=*"logic.AuthorizeUser"*%>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<title>Авторизація</title>

</head>

<body>

<h1>Вхід в систему</h1>

<%

String value = (String) session.getAttribute("ErrorLogIn");

**if** (value != **null** && value.equals("TRUE") && !AuthorizeUser.isAuthorizeUser()) {

out.println("<h1><font color=\"red\">Неправельний пароль або логін або рівеь доступу не підходить</font></h1>");

session.removeAttribute("ErrorLogIn");

}

%>

<form action=*"LogIn"* method=*"post"*>

<table cellpadding=*"3pt"*>

<tr>

<td>Ім'я користувача :</td>

<td><input type=*"text"* name=*"userName"* size=*"30"* /></td>

</tr>

<tr>

<td>Пароль :</td>

<td><input type=*"password"* name=*"password"* size=*"30"* /></td>

</tr>

</table>

<p />

<input type=*"submit"* value=*"Увійти"* />

</form>

</body>

</html>

CardList.jsp

<%@ page import=*"hibernate.Gender"*%>

<%@ page import=*"logic.OperationsWithCards"*%>

<%@ page import=*"hibernate.Card"*%>

<%@ page import=*"java.util.\*, java.text.\*"* %>

<%@ page contentType=*"text/html; charset=UTF-8"* %>

<html>

<head>

<title>

Таблиця карток

</title>

</head>

<body>

<form action=*"LogOutServlet"* method=*"post"*>

<input type=*"submit"* value=*"Вийти"* >

</form>

<a href=*"StartPage.jsp"*><button>Головна сторінка</button></a>

<% request.setCharacterEncoding("utf8");

**if** (request.getParameter("ButtonDelete") != **null**) {

Card card = **new** Card();

card.setId(Integer.valueOf(request.getParameter("ButtonDelete")));

OperationsWithCards.getOperationWithCard().deleteById(card);

}

**if** (request.getParameter("ButtonModify") != **null**) {

request.getSession().setAttribute("ButtonModify",

Integer.valueOf(request.getParameter("ButtonModify")));

response.sendRedirect("ModifyCard.jsp");

}

%>

<h1>Таблиця карток</h1>

<form name=*"CardListForm"* method=*"post"*>

<table border=*"1"* cellpadding=*"8"*>

<tr>

<% **if** (request.getParameter("cardName") != **null**) {

out.print("<th><input type=\"text\" name=\"cardName\" value=\"" +

request.getParameter("cardName") + "\" size=\"20\" /></th>");

out.print("<th>Чоловік<input type=\"checkbox\" name=\"genderMan\" " +

"value=\"" + request.getParameter("cardGender") +

"\" size=\"1\" />");

out.print("<input type=\"checkbox\" name=\"genderWoman\" " +

"value=\"" + request.getParameter("cardGender") +

"\" size=\"1\" />Жінка</th>");

out.print("<th>Н:<input type=\"text\" name=\"cardLowAgeLevel\"" +

" value=\"" + request.getParameter("cardLowAgeLevel") +

"\" size=\"1\" />");

out.print(" В:<input type=\"text\" name=\"cardHightAgeLevel\" " +

"value=\"" + request.getParameter("cardHightAgeLevel") +

"\" size=\"1\" /></th>");

out.print("<th>Так<input type=\"checkbox\" name=\"cardIsAgainTrue\" " +

"value=\"" + request.getParameter("cardIsAgain") +

"\" size=\"1\" />");

out.print("<input type=\"checkbox\" name=\"cardIsAgainFalse\" " +

"value=\"" + request.getParameter("cardIsAgain") +

"\" size=\"1\" />Ні</th>");

}

**else** {

out.println("<th><input type=\"text\" name=\"cardName\" size=" +

"\"20\" /></th>");

out.print("<th>Чоловік<input type=\"checkbox\" name=\"genderMan\" " +

"value=\"\" size=\"1\" />");

out.print("<input type=\"checkbox\" name=\"genderWoman\" " +

"value=\"\" size=\"1\" />Жінка</th>");

out.println("<th>Н:<input type=\"text\" name=\"cardLowAgeLevel\"" +

" size=\"1\" value=\"0\"/>");

out.println("В:<input type=\"text\" name=\"cardHightAgeLevel\"" +

" size=\"1\" value=\"200\"/></th>");

out.print("<th>Так<input type=\"checkbox\" name=\"cardIsAgainTrue\" " +

"value=\"\" size=\"1\" />");

out.print("<input type=\"checkbox\" name=\"cardIsAgainFalse\" " +

"value=\"\" size=\"1\" />Ні</th>");

}

%>

<th><input type=*"submit"* name=*"submit"* value=*"Використати фільтр"* size=*"15"* /></th>

</tr>

<tr>

<th>Ім'я пацієнта</th>

<th>Стать пацієнта</th>

<th>Вік пацієнта</th>

<th>Вперше</th>

</tr>

</form>

<form name=*"ModifyCard"* method=*"post"*>

<% ArrayList<Card> cards;

**if** (request.getParameter("cardName") != **null**) {

cards = OperationsWithCards.getOperationWithCard().

getAllCardsFiltr(request.getParameter("cardName"),

request.getParameter("genderMan") == **null** ?

request.getParameter("genderWoman") == **null** ?

**null** : Gender.female : request.getParameter(

"genderWoman") == **null** ? Gender.male : **null**,

Short.valueOf(request.getParameter("cardLowAgeLevel")),

Short.valueOf(request.getParameter("cardHightAgeLevel")),

request.getParameter("cardIsAgainFalse") == **null** ?

request.getParameter("cardIsAgainTrue") == **null** ?

**null** : **true** : request.getParameter(

"cardIsAgainTrue") == **null** ? **false** : **null**);

}

**else** {

cards = OperationsWithCards.getOperationWithCard().

getAllObj();

}

**for**(**int** i = 0; i < cards.size(); ++i){

out.println("<tr>");

out.println("<td>" + cards.get(i).getName() + "</td>");

out.println("<td>" + cards.get(i).getSex().getGenderName() + "</td>");

out.println("<td>" + cards.get(i).getAge() + "</td>");

out.println("<td>" + cards.get(i).getIsAgain() + "</td>");

out.println("<td><button name=\"ButtonDelete\" value=\"" +

cards.get(i).getId() + "\">Видалити</button>" +

"<button name=\"ButtonModify\" value=\"" +

cards.get(i).getId() + "\">Редагувати</button></td>");

out.println("</tr>");

} %>

</form>

</table>

</body>

</html>

Disease.jsp

<%@page import=*"hibernate.Disease"*%>

<%@page import=*"java.util.ArrayList"*%>

<%@page import=*"logic.OperationsWithDiseases"*%>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<title>Diseases</title>

</head>

<body>

<h1>Додати хворобу</h1>

<% request.setCharacterEncoding("utf8");

**if** (request.getParameter("SaveDisease") != **null**) {

Disease disease = **new** Disease();

disease.setName(request.getParameter("SaveDisease"));

OperationsWithDiseases.getOperationsWithDiseases().register(disease);

}

**if** (request.getParameter("DeleteDisease") != **null**) {

Disease disease = **new** Disease();

disease.setId(Integer.valueOf(request.getParameter("DeleteDisease")));

OperationsWithDiseases.getOperationsWithDiseases().deleteById(disease);

}

%>

<form name=*"saveDisease"* method=*"post"*>

<table cellpadding=*"3pt"*>

<tr>

<td>Ім'я нової хвороби</td>

<td><input type=*"text"* name=*"SaveDisease"*></td>

<td><input type=*"submit"* value=*"Додати"*></td>

</tr>

</table>

</form>

<form action=*"LogOutServlet"* method=*"post"*>

<input type=*"submit"* value=*"Вийти"* >

</form>

<a href=*"StartPage.jsp"*><button>Головна сторінка</button></a>

<h1>Таблиця хвороб</h1>

<form name=*"deleteDisease"* method=*"post"*>

<table border=*"1"* cellpadding=*"8"*>

<tr>

<td>Назва хвороби</td>

</tr>

<% ArrayList<Disease> diseases = OperationsWithDiseases.

getOperationsWithDiseases().getAllObj();

**for**(**int** i = 0; i < diseases.size() ; ++i) {

out.println("<tr><td>" + diseases.get(i).getName() + "</td><td><button " +

"name=\"DeleteDisease\" value=\"" + diseases.get(i).getId() +

"\">Видалити</button></td></tr>");

}

%>

</table>

</form>

</body>

</html>

ModifyCard.jsp

<%@page import=*"logic.OperationsWithDiagnosis"*%>

<%@page import=*"logic.OperationsWithDiseases"*%>

<%@page import=*"logic.OperationWithSessions"*%>

<%@page import=*"hibernate.Gender"*%>

<%@page import=*"logic.OperationsWithCards"*%>

<%@page import=*"java.util.ArrayList"*%>

<%@page import=*"hibernate.Note"*%>

<%@page import=*"logic.OperationWithNotes"*%>

<%@page import=*"java.sql.Date"*%>

<%@page import=*"hibernate.Session"*%>

<%@page import=*"hibernate.Disease"*%>

<%@page import=*"hibernate.Diagnos"*%>

<%@ page import=*"hibernate.Card"*%>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<title>Редагування картки</title>

</head>

<body>

<%! Card card;%>

<% request.setCharacterEncoding("utf8");

card = **new** Card();

**if** (request.getParameter("ButtonSessionsDelete") != **null**) {

Session sess = **new** Session();

sess.setId(Integer.valueOf(request.getParameter("ButtonSessionsDelete")));

OperationWithSessions.getOperationWithSessions().deleteById(sess);

}

**if** (request.getParameter("ButtonNoteDelete") != **null**) {

Note note = **new** Note();

note.setId(Integer.valueOf(request.getParameter("ButtonNoteDelete")));

OperationWithNotes.getOperationWithNotes().deleteById(note);

}

card.setId((Integer) request.getSession().getAttribute("ButtonModify"));

**if** ( request.getParameter("ButtonSave") != **null**) {

card.setAge(Short.valueOf(request.getParameter("CardAge")));

card.setIsAgain((request.getParameter("CardIsAgain") != **null** ? **true** :**false**));

card.setName(request.getParameter("CardName"));

card.setNote(request.getParameter("CardNote"));

card.setSex(Gender.getValue(request.getParameter("CardGender")));

**if**( OperationsWithCards.getOperationWithCard().update(card))

out.print("<h1>Успішно</h1>");

**else** out.print("<h1>Помилка</h1>");

}

card = OperationsWithCards.getOperationWithCard().isRegisted(card);

%>

<h1>Картка</h1>

<form name=*"CardDate"* method=*"post"*>

<table>

<tr><td>Ім'я:</td><td><input type=*"text"* name=*"CardName"*

value=*"*<%=card.getName() %>*"*></td></tr>

<tr><td>Вік:</td><td><input type=*"text"* name=*"CardAge"*

value=*"*<%=card.getAge() %>*"*></td></tr>

<tr><td>Стать:</td><td>Чоловік<input type=*"radio"* name=*"CardGender"* value=*"Man"*

<%=card.getSex().equals(Gender.male) ? "checked" : "" %>>

<input type=*"radio"* name=*"CardGender"* value=*"Woman"*

<%=card.getSex().equals(Gender.female) ? "checked" : "" %>>Жінка</td></tr>

<tr><td>Вперше:</td><td><input type=*"checkbox"* name=*"CardIsAgain"*

value=*"*<%=card.getIsAgain() %>*"*

<%=card.getIsAgain().equals(**true**) ? "checked" : "" %>></td></tr>

<tr><td>Нотаток:</td><td><textarea cols=*"40"* rows=*"5"* name=*"CardNote"*><%=card.getNote()%></textarea></td></tr>

</table>

<button name=*"ButtonSave"* value=*"Save"*>Зберегти</button>

<button name=*"ButtonCencel"* value=*"Cancel"*>Відмінити</button>

</form>

<br>

<form action=*"LogOutServlet"* method=*"post"*>

<input type=*"submit"* value=*"Вийти"* >

</form>

<a href=*"StartPage.jsp"*><button>Головна сторінка</button></a>

<%

out.println("<h1>Нотатки</h1>");

ArrayList<Note> notes = card.getAllNotes();

**for**(**int** i = 0; i < notes.size(); ++i) {

**if** ( request.getParameter("ButtonNoteSave") != **null** &&

Integer.valueOf(request.getParameter("ButtonNoteSave")).

equals(notes.get(i).getId())) {

notes.get(i).setHidden\_note(request.getParameter("userNote"));

notes.get(i).setHide(request.getParameter("userNoteIsHide") !=

**null** ? **true** : **false**);

notes.get(i).setDate(Date.valueOf(request.getParameter("userNoteDate")));

OperationWithNotes.getOperationWithNotes().update(notes.get(i));

}

out.println(

"<form name=\"Note #" + (1 + i) + "\" method=\"post\">" +

"<h2>Нотаток " + (i + 1) +

"</h2><textarea cols=\"40\" rows=\"3\" name=\"userNote\">" +

notes.get(i).getHidden\_note() + "</textarea>" +

"<br><button value=\"" + notes.get(i).getId() +

"\" name=\"ButtonNoteSave\" >Зберегти" +

"</button><button value=\"" + notes.get(i).getId() +

"\" name=\"ButtonNoteDelete\" >Видалити" +

"</button><button value=\"" + notes.get(i).getId() +

"\" name=\"ButtonNoteCencel\" >Відмінити</button>" +

"Hide note <input type=\"checkbox\" name=\"userNoteIsHide\" " +

"value=\"" + notes.get(i).getHide() +

"\" size=\"1\"" + (notes.get(i).getHide().equals(**true**) ? "checked" : "") +

">Час <input type=\"text\" name=\"userNoteDate\"" +

"size=\"5\" value=\"" + notes.get(i).getDate().toString() + "\">" +

"<input type=\"hidden\" name=\"ButtonModify\" value=\"" +

card.getId() + "\"></form><br>");

}

out.println("<h1>Сесії лікування</h1>");

ArrayList<Session> sessions = card.getAllSessions();

**for**(**int** i = 0; i < sessions.size(); ++i) {

**if** ( request.getParameter("ButtonSessionsSave") != **null** &&

Integer.valueOf(request.getParameter("ButtonSessionsSave")).

equals(sessions.get(i).getId())) {

sessions.get(i).setResult(request.getParameter("sessionResult") !=

**null** ? **true** : **false** );

Diagnos diagnos = sessions.get(i).getDiagnos();

diagnos.setDescription(request.getParameter("sessionDiagnosDescription"));

Disease disease = **new** Disease();

disease.setName(request.getParameter("sessionDiagnosDescriptionDisease"));

**if** (disease.getName() != **null**) {

disease = OperationsWithDiseases.getOperationsWithDiseases().isRegisted(disease);

diagnos.setDisease(disease);

}

OperationsWithDiagnosis.getOperationsWithDiagnosis().update(diagnos);

OperationWithSessions.getOperationWithSessions().update(sessions.get(i));

}

out.println(

"<form name=\"Session #" + (1 + i) + "\" method=\"post\">" +

"<h2>Сесія " + (i + 1) +

"</h2><textarea cols=\"40\" rows=\"3\" name=\"sessionDiagnosDescription\">" +

sessions.get(i).getDiagnos().getDescription() + "</textarea>" +

"<br><button value=\"" + sessions.get(i).getId() +

"\" name=\"ButtonSessionsSave\" >Зберегти" +

"</button><button value=\"" + sessions.get(i).getId() +

"\" name=\"ButtonSessionsDelete\" >Видалити" +

"</button><button value=\"" + sessions.get(i).getId() +

"\" name=\"ButtonSessionsCencel\" >Відмінити</button>" +

"Результат лікування <input type=\"checkbox\" name=\"sessionResult\" " +

"value=\"" + sessions.get(i).getResult() +

"\" size=\"1\"" + (sessions.get(i).getResult().equals(**true**) ?

"checked" : "") + "/><select size=\"1\" " +

"name=\"sessionDiagnosDescriptionDisease\"><option disabled selected>" +

sessions.get(i).

getDiagnos().getDisease().getName() + "</option>");

ArrayList<Disease> diseases = OperationsWithDiseases.getOperationsWithDiseases().

getAllObj();

**for**( i = 0; i < diseases.size(); ++i)

out.println("<option value=\"" + diseases.get(i).getName() + "\">" +

diseases.get(i).getName() + "</option>");

out.println("</select><input type=\"hidden\" name=\"ButtonModify\" value=\"" +

card.getId() + "\"></form><br>");

}

%>

</body>

</html>

ModifyUser.jsp

<%@page import=*"logic.OperationWithNotes"*%>

<%@ page import=*"hibernate.User"*%>

<%@ page import=*"hibernate.Note"*%>

<%@ page import=*"logic.OperationsWithUsers"*%>

<%@ page import=*"java.util.ArrayList"*%>

<%@ page import=*"java.sql.Date"*%>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<title>Редагування користовача</title>

</head>

<body>

<%! User user;%>

<% request.setCharacterEncoding("utf8");

user = **new** User("", **null**, "", "");

**if**

(request.getParameter("ButtonCardOpen") != **null**) {

request.getSession().setAttribute("ButtonModify",

Integer.valueOf(request.getParameter("ButtonCardOpen")));

response.sendRedirect("ModifyCard.jsp");

}

**if** (request.getParameter("Save") != **null**) {

**boolean** status;

user = **new** User(request.getParameter("userName"),

Short.valueOf(request.getParameter("userAccess\_level")),

request.getParameter("userPhone"), request.getParameter("userPwd"));

**if** (!request.getParameter("UserId").equals("null")) {

user.setId(Integer.valueOf(request.getParameter("UserId")));

request.setAttribute("ButtonModify", Integer.valueOf(request.getParameter("UserId")));

status = OperationsWithUsers.getOperationWithUsers().update(user);

} **else** status = OperationsWithUsers.getOperationWithUsers().register(user);

**if**( status) {

out.print("<h1>Успішно</h1>");

request.getSession().setAttribute("ButtonModify", user.getId());

}

**else** out.print("<h1>Помилка</h1>");

}

**if**( request.getParameter("UserAdd") != **null**) {

out.print("<h1>Додати нового користувача</h1>");

}

**else** {

System.out.println(request.getParameter("ButtonModify"));

**if** (request.getParameter("ButtonModify") != **null** ||

request.getSession().getAttribute("ButtonModify") != **null**) {

**if** (request.getParameter("ButtonModify") != **null**)

request.setAttribute("ButtonModify",

Integer.valueOf(request.getParameter("ButtonModify")));

**else** request.setAttribute("ButtonModify",

request.getSession().getAttribute("ButtonModify"));

request.getSession().setAttribute("ButtonModify",

request.getAttribute("ButtonModify"));

}

out.print("<h1>Редагувати користувача</h1>");

user.setId((Integer) request.getAttribute("ButtonModify"));

user = OperationsWithUsers.getOperationWithUsers().isRegisted(user);

}

**if** (request.getParameter("ButtonNoteDelete") != **null**) {

Note note = **new** Note();

note.setId(Integer.valueOf(request.getParameter("ButtonNoteDelete")));

OperationWithNotes.getOperationWithNotes().deleteById(note);

}

%>

<form name=*"AddModify"* method=*"post"*>

<table cellpadding=*"3pt"*>

<tr>

<td>Ім'я користувача :</td>

<td><input type=*"text"* name=*"userName"* value=*"*<%=user.getName()%>*"*

size=*"30"* /></td>

</tr>

<tr>

<td>Телефон користувача :</td>

<td><input type=*"text"* name=*"userPhone"* value=*"*<%=user.getPhone()%>*"*

size=*"30"* /></td>

</tr>

<tr>

<td>Рівень доступу користувача :</td>

<td><input type=*"text"* name=*"userAccess\_level"*

value=*"*<%**if** (user.getAccess\_level() == **null**) out.print("");

**else** out.print(user.getAccess\_level());%>*"* size=*"30"* /></td>

</tr>

<tr>

<td>Пароль користувача :</td>

<td><input type=*"password"* name=*"userPwd"* size=*"30"* /></td>

</tr>

</table>

<p />

<input type=*"hidden"* name=*"UserId"* value=*"*<%=user.getId()%>*"* />

<input type=*"hidden"* name=*"Save"* value=*"1"* />

<input type=*"submit"* value=*"Зберегти"* />

</form>

<form action=*"LogOutServlet"* method=*"post"*>

<input type=*"submit"* value=*"Вийти"* >

</form>

<a href=*"StartPage.jsp"*><button>Головна сторінка</button></a>

<% **if** (request.getParameter("UserAdd") == **null**) {

out.println("<h1>Нотатки</h1>");

ArrayList<Note> notes = user.getAllNotes();

**for**(**int** i = 0; i < notes.size(); ++i) {

**if** ( request.getParameter("ButtonNoteSave") != **null** &&

Integer.valueOf(request.getParameter("ButtonNoteSave")).

equals(notes.get(i).getId())) {

notes.get(i).setHidden\_note(request.getParameter("userNote"));

notes.get(i).setHide(request.getParameter("userNoteIsHide") !=

**null** ? **true** : **false**);

notes.get(i).setDate(Date.valueOf(request.getParameter("userNoteDate")));

OperationWithNotes.getOperationWithNotes().update(notes.get(i));

}

out.println(

"<form name=\"Note #" + (1 + i) + "\" method=\"post\">" +

"<h2>Нотаток " + (i + 1) +

" Ім'я картки <button value=\"" + notes.get(i).getCard().getId() +

"\" name=\"ButtonCardOpen\" >" + notes.get(i).getCard().getName()

+ "</button></h2>" +

"<textarea cols=\"40\" rows=\"3\" name=\"userNote\">" +

notes.get(i).getHidden\_note() + "</textarea>" +

"<br><button value=\"" + notes.get(i).getId() +

"\" name=\"ButtonNoteSave\" >Зберегти" +

"</button><button value=\"" + notes.get(i).getId() +

"\" name=\"ButtonNoteDelete\" >Видаоити" +

"</button><button value=\"" + notes.get(i).getId() +

"\" name=\"ButtonNoteCencel\" >Відмінити</button>" +

"Hide note <input type=\"checkbox\" name=\"userNoteIsHide\" " +

"value=\"" + notes.get(i).getHide() +

"\" size=\"1\"" + (notes.get(i).getHide().equals(**true**) ?

"checked" : "") + "/>Час <input type=\"text\" name=\"userNoteDate\"" +

"size=\"5\" value=\"" + notes.get(i).getDate().toString() + "\">" +

"<input type=\"hidden\" name=\"ButtonModify\" value=\"" +

user.getId() + "\"></form><br>");

}

}

%>

</body>

</html>

StartPage.jsp

<%@page import=*"logic.AuthorizeUser"*%>

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<title>Головна сторінка</title>

</head>

<body>

<h2>Головна сторінка Адміністратора</h2>

<table>

<tr>

<td><a href=*"UserList.jsp"*><button>Список користувачів</button></a></td>

</tr>

<tr>

<td><a href=*"CardList.jsp"*><button>Список карток пацієнтів</button></a></td>

</tr>

<tr>

<td><form action=*"ModifyUser.jsp"* method=*"post"*>

<input type=*"hidden"* name=*"UserAdd"* value=*"True"*>

<input type=*"submit"* value=*"Додати користувача"*>

</form>

</td>

</tr>

<tr>

<td><a href=*"Disease.jsp"*><button>Додати хворобу</button></a></td>

</tr>

<tr>

<td><form action=*"ModifyUser.jsp"* method=*"post"*>

<button name=*"ButtonModify"* value=

*"*<%=AuthorizeUser.getAuthorizeUser().getId()%>*"*>Налаштування акаунту</button>

</form>

</td>

</tr>

<tr>

<td><a href=*"NewFile.jsp"*><button>Перейти в режим доктора(Віталін проект)</button></a></td>

</tr>

</table>

<form action=*"LogOutServlet"* method=*"post"*>

<input type=*"submit"* value=*"Вийти"* >

</form>

</body>

</html>

UserList.jsp

<%@ page import=*"logic.AuthorizeUser"*%>

<%@ page import=*"logic.OperationsWithUsers"*%>

<%@ page import=*"hibernate.User"*%>

<%@ page import=*"java.util.\*, java.text.\*"* %>

<%@ page contentType=*"text/html; charset=UTF-8"* %>

<html>

<head>

<title>

Таблиця користувачів

</title>

</head>

<body>

<form action=*"LogOutServlet"* method=*"post"*>

<input type=*"submit"* value=*"Вийти"* >

</form>

<a href=*"StartPage.jsp"*><button>Головна сторінка</button></a>

<% request.setCharacterEncoding("utf8");

**if** (request.getParameter("ButtonDelete") != **null**) {

User user = **new** User();

user.setId(Integer.valueOf(request.getParameter("ButtonDelete")));

OperationsWithUsers.getOperationWithUsers().deleteById(user);

}

**if** (request.getParameter("ButtonModify") != **null**) {

request.getSession().setAttribute("ButtonModify", Integer.

valueOf(request.getParameter("ButtonModify")));

response.sendRedirect("ModifyUser.jsp");

}

%>

<h1>Таблиця користувачів</h1>

<table border=*"1"* cellpadding=*"8"*>

<form name=*"UserListForm"* method=*"post"*>

<tr>

<% **if** (request.getParameter("userName") != **null**) {

out.print("<th><input type=\"text\" name=\"userName\" value=\"" +

request.getParameter("userName") + "\" size=\"20\" /></th>");

out.print("<th><input type=\"text\" name=\"userPhone\" value=\"" +

request.getParameter("userPhone") + "\"size=\"20\" /></th>");

out.print("<th>L:<input type=\"text\" name=\"userLowAccessLevel\"" +

" value=\"" +

request.getParameter("userLowAccessLevel") +

"\" size=\"1\" />");

out.print(" H:<input type=\"text\" name=\"userHightAccessLevel\" " +

"value=\"" + request.getParameter("userHightAccessLevel") +

"\" size=\"1\" /></th>");

}

**else** {

out.println("<th><input type=\"text\" name=\"userName\" size=" +

"\"20\" /></th>");

out.println("<th><input type=\"text\" name=\"userPhone\" size=" +

"\"20\" /></th>");

out.println("<th>L:<input type=\"text\" name=\"userLowAcces" +

"sLevel\" size=\"1\" value=\"0\"/>");

out.println("H:<input type=\"text\" name=\"userHightAccessLev" +

"el\" size=\"1\" value=\"2\"/></th>");

}

%>

<th><input type=*"submit"* name=*"submit"* value=*"Використати фільтр"* size=*"15"* /></th>

</tr>

</form>

<tr>

<th>Ім'я</th>

<th>Телефон</th>

<th>Уравінь доступу</th>

</tr>

<form name=*"ModifyUser"* method=*"post"*>

<% ArrayList<User> users;

**if** (request.getParameter("userName") != **null**) {

users = OperationsWithUsers.getOperationWithUsers().

getAllObjSatisfyFiltr(request.getParameter("userName"),

request.getParameter("userPhone"),

Short.valueOf(request.getParameter("userLowAccessLevel")),

Short.valueOf(request.getParameter("userHightAccessLevel")));

}

**else** {

users = OperationsWithUsers.getOperationWithUsers().

getAllObj();

}

**for**(**int** i = 0; i < users.size(); ++i){

out.println("<tr>");

out.println("<td>" + users.get(i).getName() + "</td>");

out.println("<td>" + users.get(i).getPhone() + "</td>");

out.println("<td>" + users.get(i).getAccess\_level() + "</td>");

**if**( users.get(i).equals((Object) AuthorizeUser.getAuthorizeUser()))

out.println("<td>Це Ви</td>");

**else** out.println("<td><button name=\"ButtonDelete\" value=\"" +

users.get(i).getId() + "\">Видалити</button>" +

"<button name=\"ButtonModify\" value=\"" +

users.get(i).getId() + "\">Редагувати</button>" +

"<button name=\"ButtonenterLikeThisUser\" value=\"" +

users.get(i).getId() + "\"><font color=\"blue\">Увійти" +

" як цей користувач</font></button></td>");

out.println("</tr>");

} %>

</form>

</table>

</body>

</html>